PB165 – Grafy a sítě

11. Zajímavé grafy a peer to peer sítě

Obsah přednášky

- Náhodné grafy
- Náhodné geometrické grafy
- Power-law sítě
- Peer to peer sítě
  - Prohledávání v nestrukturovaných sítích
  - Strukturované sítě
  - Chord

Náhodné grafy (Erdös-Renyi)

- Pravděpodobnostní modely grafů

Definice

$G(n,p)$ model náhodného grafu: Mějme graf s n vrcholy, pak neorientovaná hrana vznikne mezi dvojicí vrcholů s pravděpodobností p, nezávisle na ostatních dvojicích uzlů.

-
- Pravděpodobnost, že graf G má k hran je dána výrazem

$$p^k(1-p)^{\binom{n}{2}-k}$$

- Pravděpodobnost, že graf $G \in G(n,p)$ obsahuje k hran má binomiální rozložení

$$\binom{\binom{n}{2}}{k} p^k(1-p)^{\binom{n}{2}-k}$$

Prahy náhodných grafů

Věta

Mějme model náhodného grafu $G(n,p)$ a nechť $p = c\frac{\log n}{n}$. Je-li $c > 1$, pak prakticky všechny grafy nemají izolované (nespojené) vrcholy. Je-li $c < 1$, pak prakticky všechny grafy mají alespoň jeden izolovaný vrchol.

Důkaz si pouze naznačíme:

- Nechť X je počet izolovaných vrcholů v grafu $G \in G(n,p)$.
- Pak *horní hranice* $E[X] = n(1-p)^{(n-1)} \approx n^{(1-c)}$
- Pravděpodobnost, že $X = 0$ je přibližně $\frac{1}{E[X]}$.
- $E[X]$ se blíží 0, pokud $c > 1$, tedy počet izolovaných vrcholů se blíží 0.
- Naopak, pro $c < 1$ se $E[X]$ blíží nekonečnu, a tedy pravděpodobnost, že graf G nemá izolovaný uzel, se blíží 0.

Náhodné geometrické grafy

Definice

Rozhoďme n bodů náhodně na jednotkovou síť. Náhodný geometrický graf $G(n, r)$ má n uzlů odpovídající n bodům a dva uzly jsou spojeny hranou pokud jsou ve vzdálenosti menší nebo rovné r. Vzdálenost mezi dvěma body $u = (x_1, y_1)$ a $v = (x_2, y_2)$ je definována jako
$d(u, v) = \max(|x_1 - x_2|, |y_1 - y_2|)$.

Jedná se o oblíbený model senzorových sítí.

Věta

Pokud $r \geq \sqrt{\frac{c \log n}{n}}$, *kde* c *je konstanta* > 4, *pak* $G(n, r)$ *je asymptoticky prakticky určitě souvislý graf, tj.* *Pravděpodobnost*($G(n, r)$ *je souvislý graf*) *se blíží 1 jak se* n *blíží nekonečnu.*

Důkaz je naznačen dále:

- Rozdělme jednotkovou síť na schránky velikosti $r/2 \times r/2$. Počet přihrádek je $4/r^2$ ($r < 1$).
- $G(n, r)$ je souvislé, pokud žádná přihrádka není prázdná.
- Nechť $r = \sqrt{\frac{c \log n}{n}}$, pak pravděpodobnost, že přihrádka je prázdná je menší nebo rovna $n^{-c/4}$.
- Nechť X je počet prázdných přihrádek. Pak $E[X] \leq n^{1-c/4}$, tj. $E[X]$ se blíží nule, pokud $c > 4$.
- Pravděpodobnost($X > 0$) $\leq E[X] \to 0$.

Power-law grafy

Definice

Power law je jakýkoliv polynomiální vztah, který nezávisí na měřítku. Pro dvě proměnné se nejčastěji vyjadřuje vztahem

$$f(x) = ax^k + o(x^k)$$

kde a a k jsou konstanty a $o(x^k)$ je asymptoticky malá (tj. zanedbatelná) funkce x.

Pro reálné sítě platí následující empirické tvrzení:
Distribuce stupně vrcholu sítě sleduje power law, tj. počet vrcholů stupně k je $ck^{-\beta}$.
Příklady:

- Internet a WWW: $\beta = 2,1$
- Sociální sítě (např. citační grafy): $\beta = 3$
- Biologické sítě (grafy interakce proteinů): $\beta = 2,5$

Většina grafů reprezentujících reálný svět má $\beta \in \langle 1, 4 \rangle$.

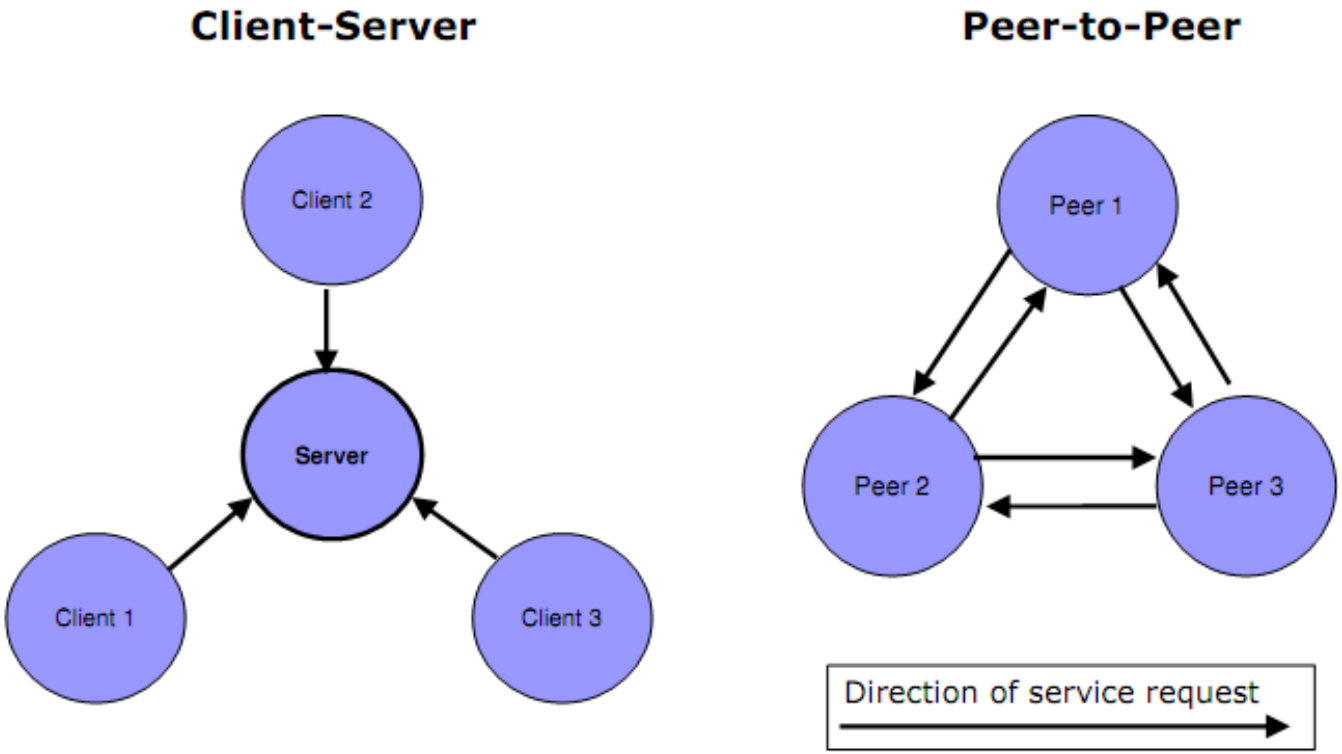
Prohledávání v nestrukturovaných sítích

Máme rozsáhlou síť (graf), každý uzel nese nějaká data. Jak najdeme hledaná data?

- Centrální index
  - Napster
  - Snadné, ale neškáluje (Google má $>$ 100.000 serverů)
- Decentralizované
  - Gnutella: Primitivní hledání: záplava
  - Chytřejší algoritmy

Základ výzkumu v oblasti *peer to peer (P2P) sítí*

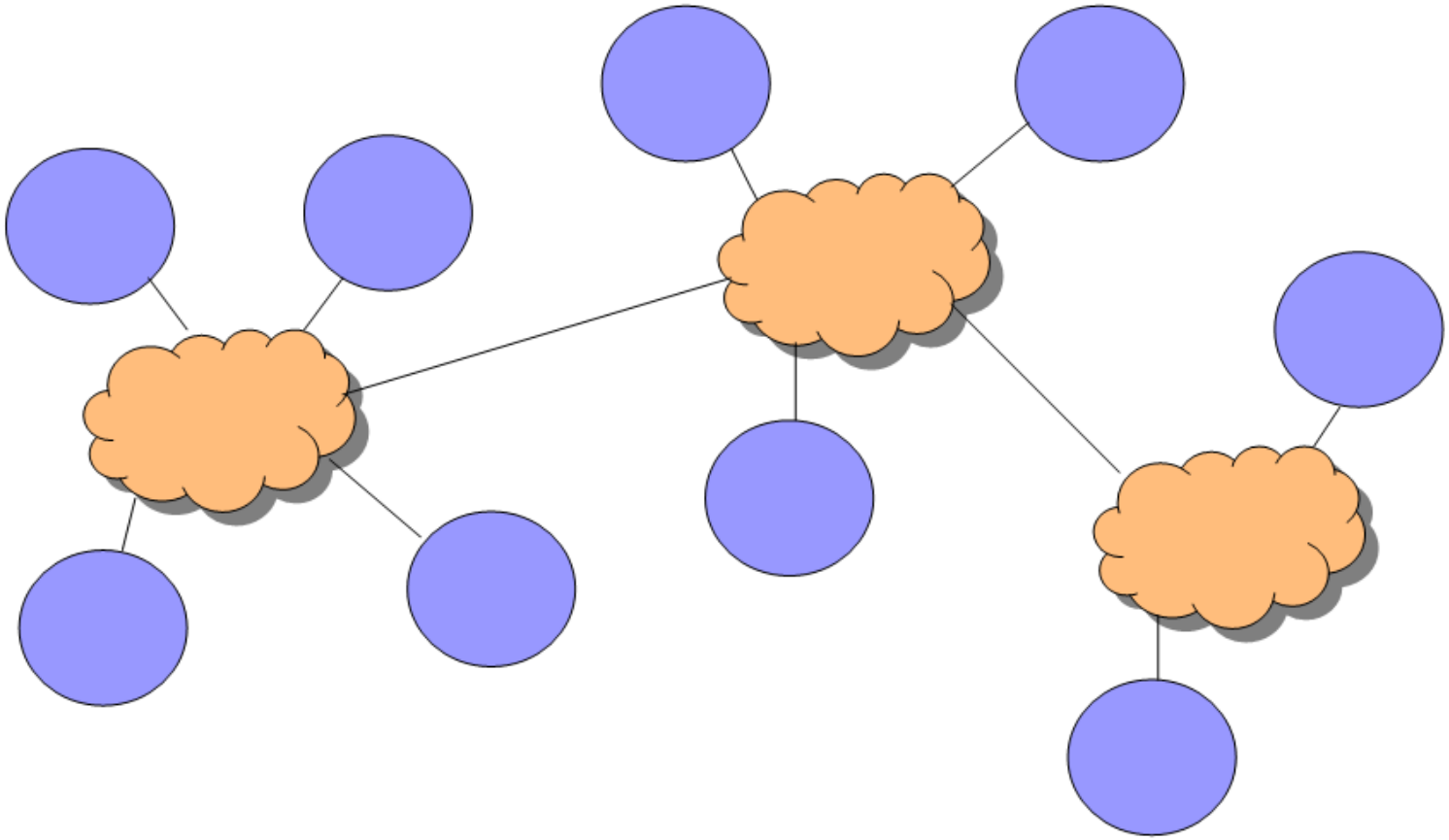
Srovnání komunikační struktury



Překryvové sítě & P2P

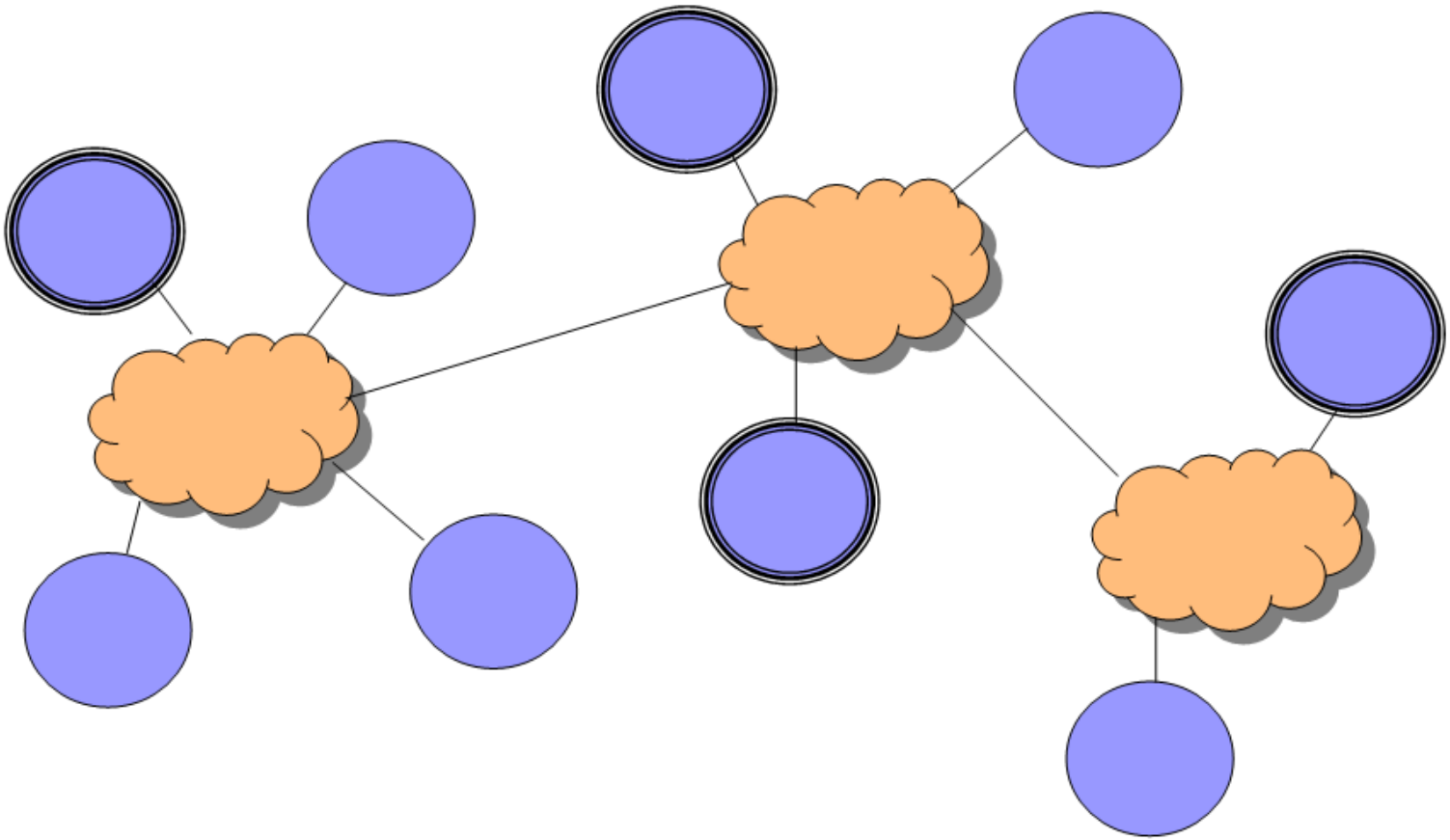
- P2P síť je typicky „virtuální“ síť utvořená nad existující síťovou infrastrukturou (např. nad sítí Internet)
  - *překryvová síť* je využita pro indexování a zjišťování sousedů (peerů) $\Rightarrow$ P2P systém je tak nezávislý na topologii (základové) fyzické sítě
  - vlastní data jsou obvykle přenášena po fyzické síti
- nový peer musí za účelem připojení se k P2P síti získat informaci o nejméně jednom jejím členovi
  - nezbytné síťové informace: IP adresa, port, atd.
  - informace o dalších peerech mohou být získány od něj

Překryvová síť



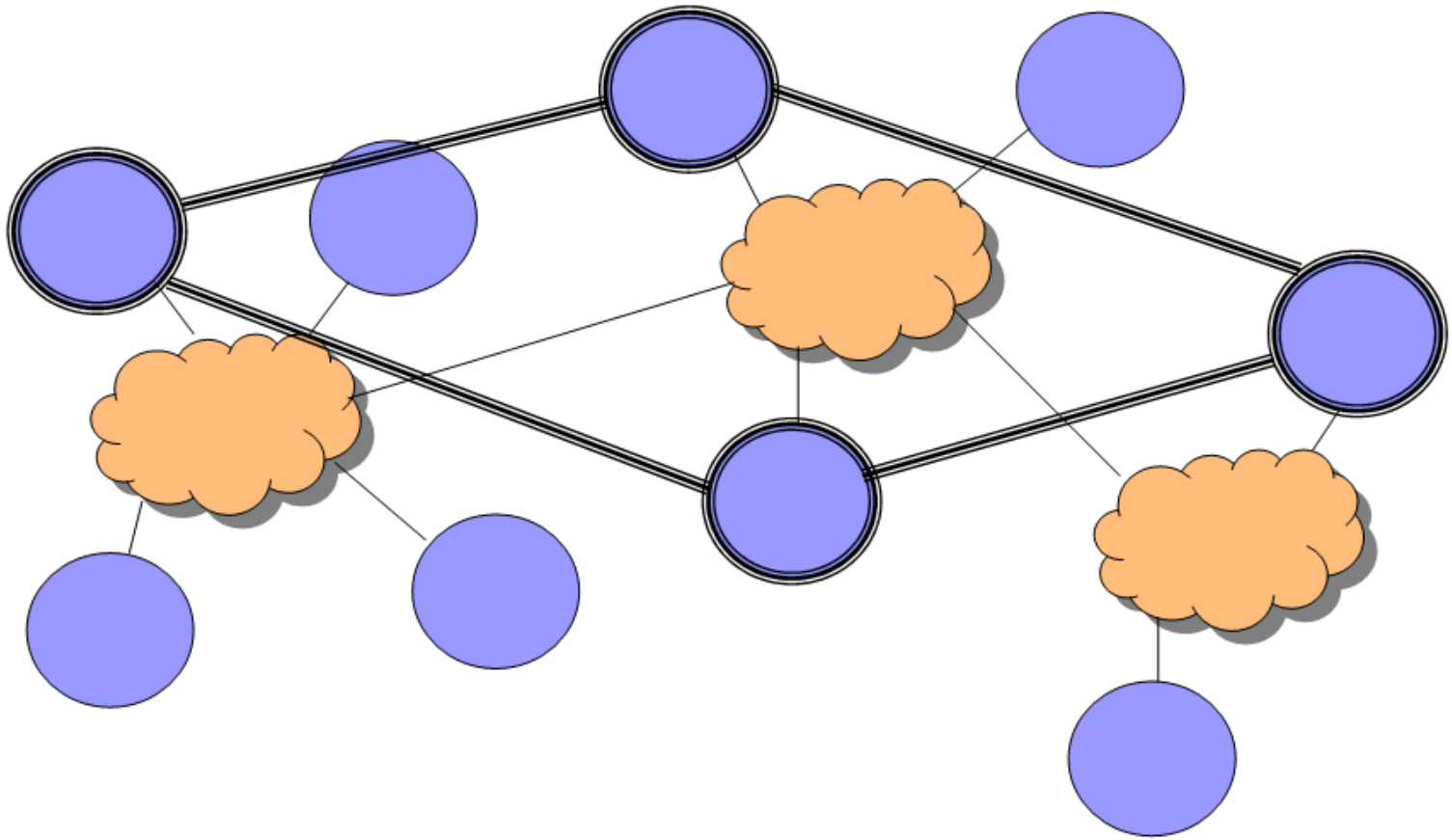
Obrázek: Překryvová vs. fyzická (základová) síť.

Překryvová síť



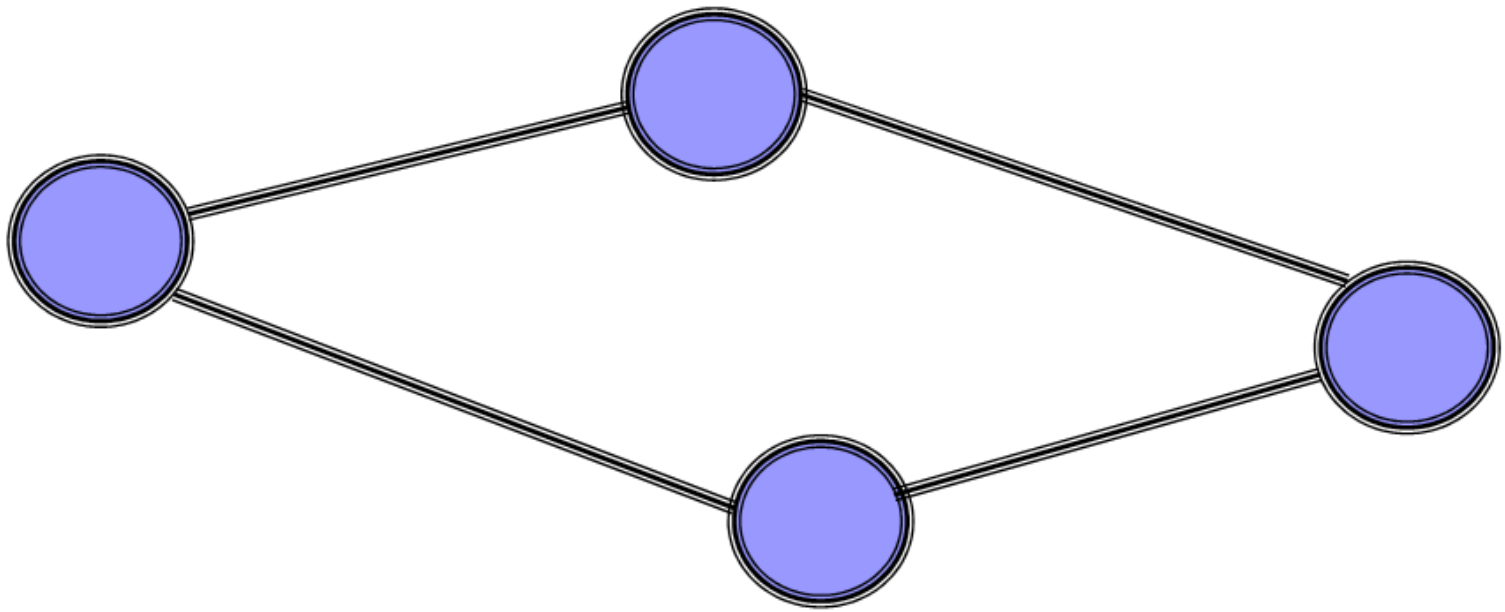
Obrázek: Překryvová vs. fyzická (základová) síť.

Překryvová síť



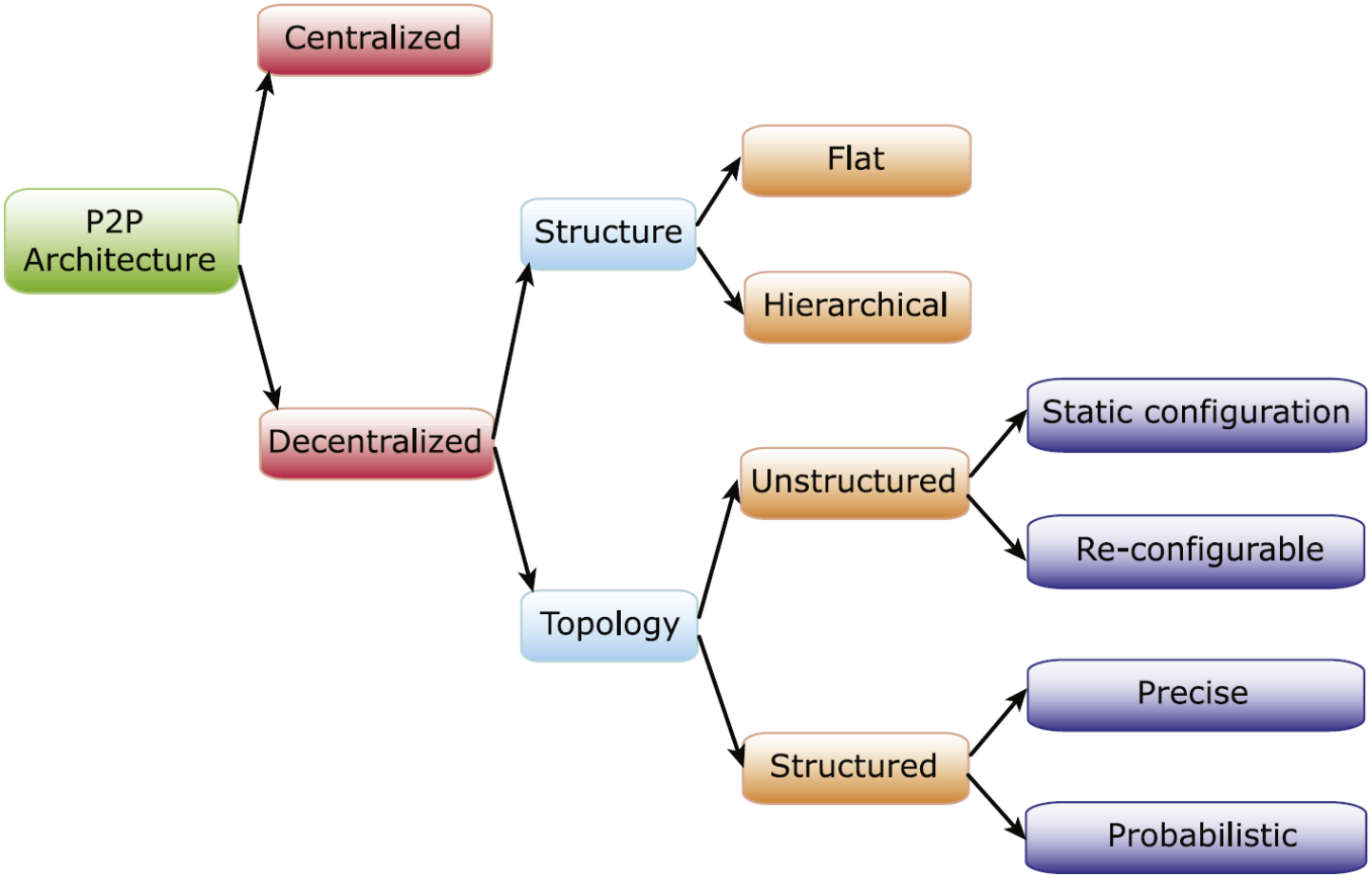
Obrázek: Překryvová vs. fyzická (základová) síť.

Překryvová síť

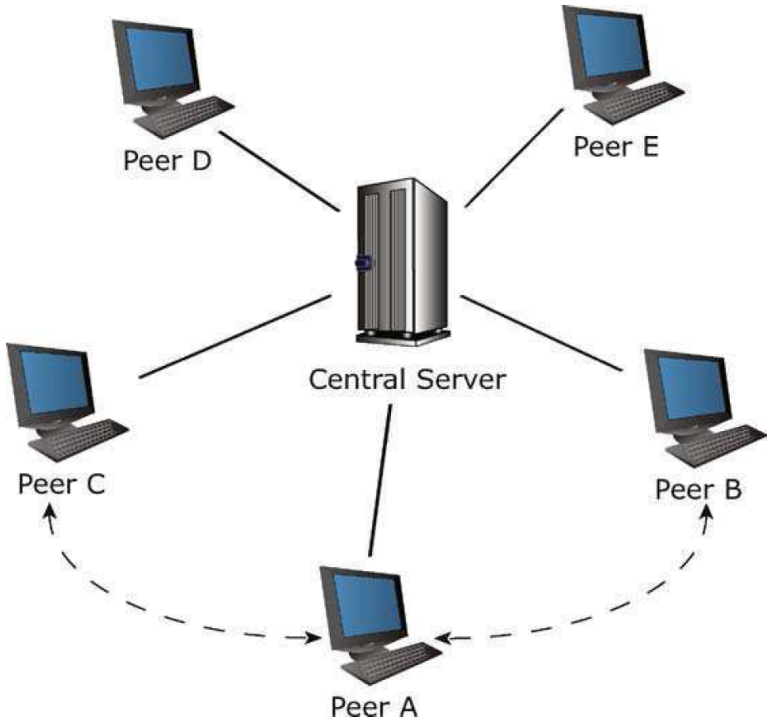


Obrázek: Překryvová vs. fyzická (základová) síť.

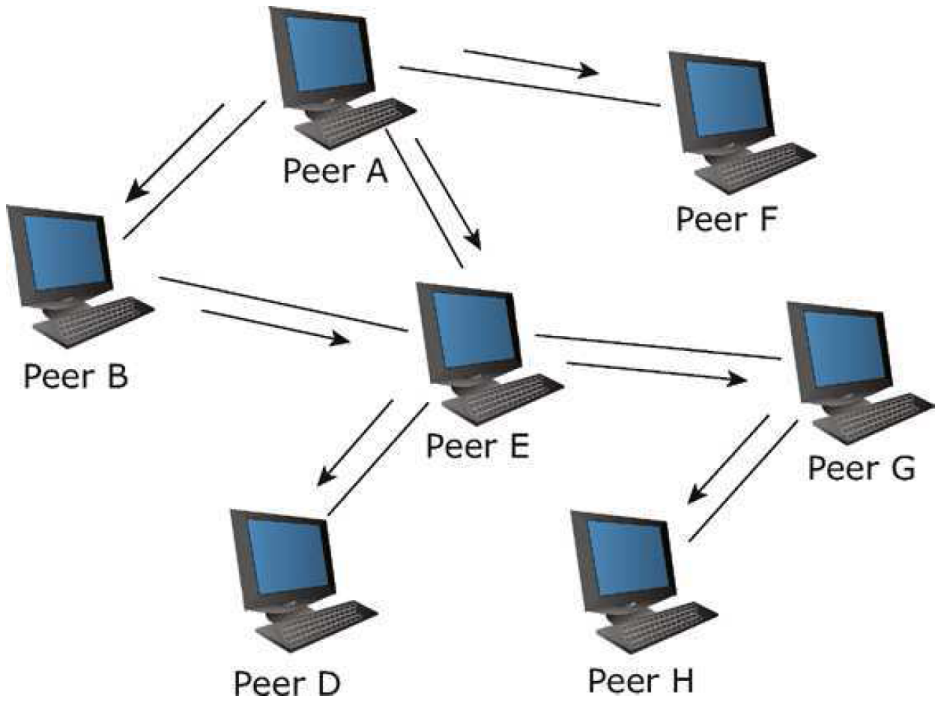
Základní členění P2P systémů



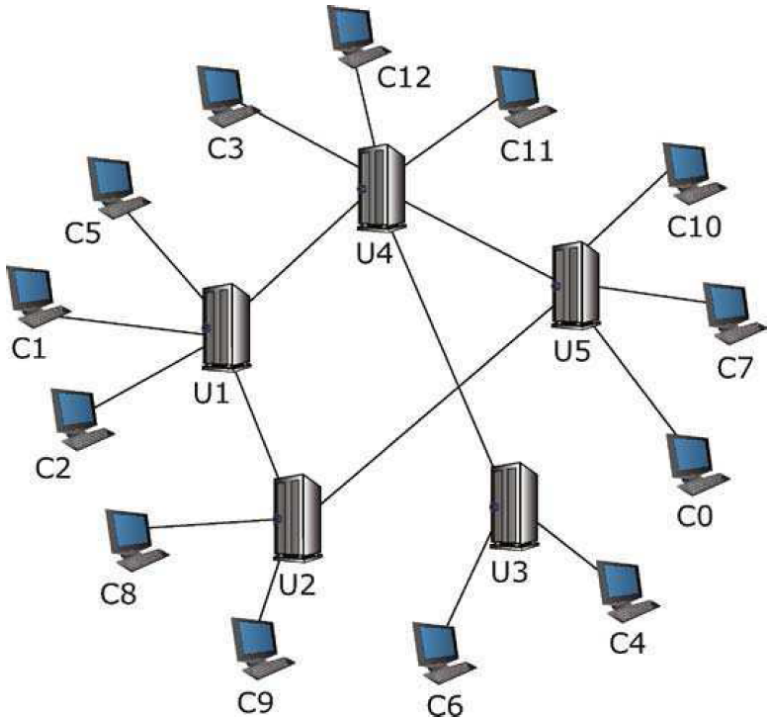
Obrázek: Základní členění P2P systémů.



Obrázek: *Centralizované P2P systémy:* Peer A zasílá dotaz na seznam uzlů mající požadovaná data centrálnímu serveru. Jakmile uzel A od centrálního serveru získá požadovaný seznam uzlů splňujících jeho požadavky (tj. např. uzly B a C), provádí vlastní komunikaci „napřímo“, tj. bez intervence centrálního serveru.



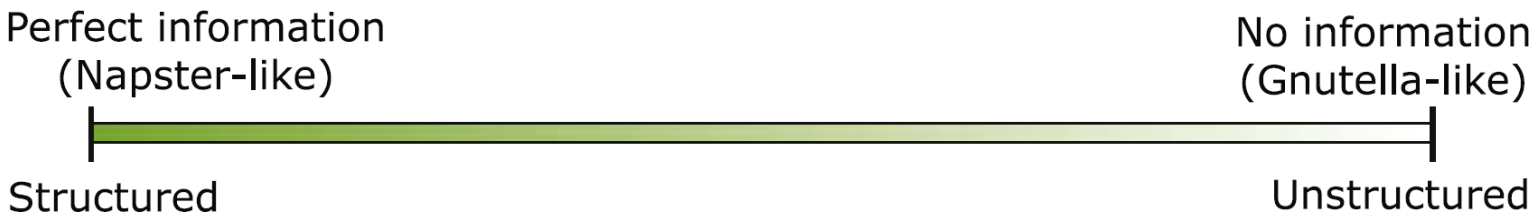
Obrázek: *Decentralizované P2P systémy:* (Peer A požaduje určitá data, která jsou uložena na peeru D a peeru H) Požadavek je broadcastován všem sousedům peera A, poté všem jejich sousedům, atd., až je doručen všem peerům v P2P síti.

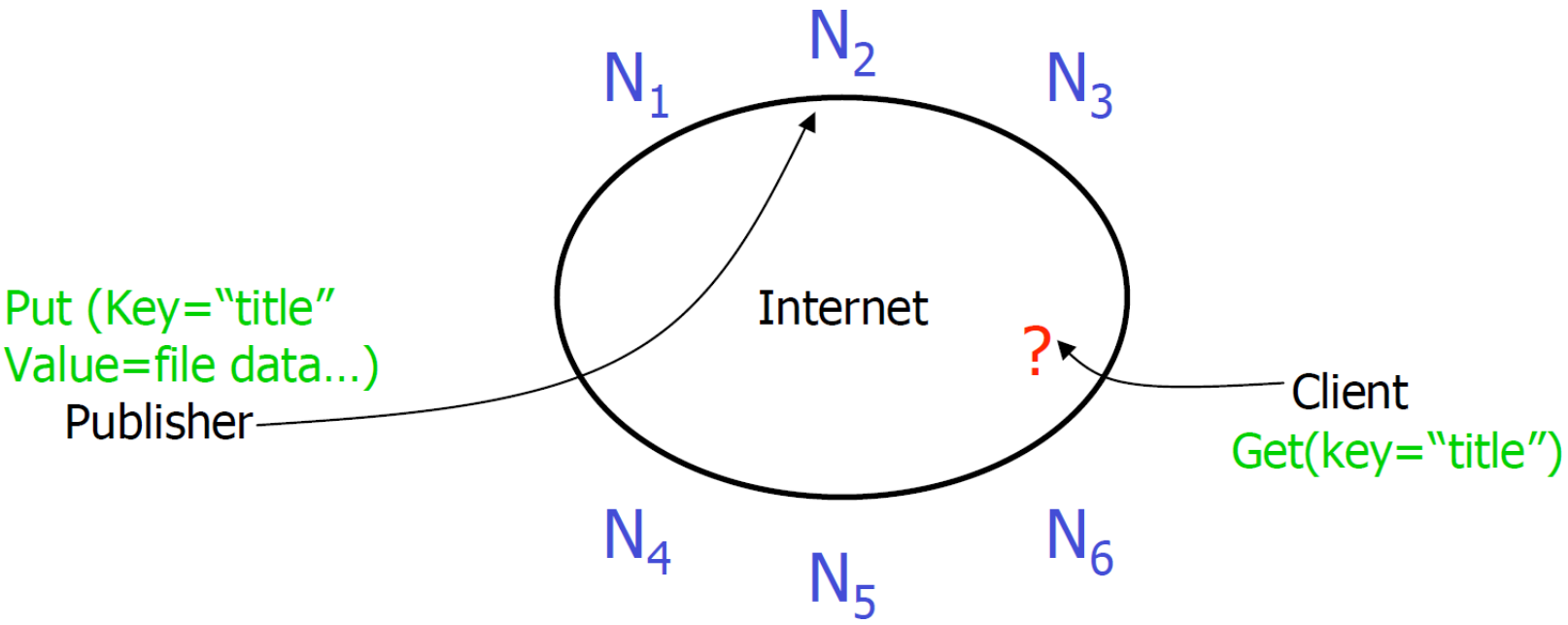


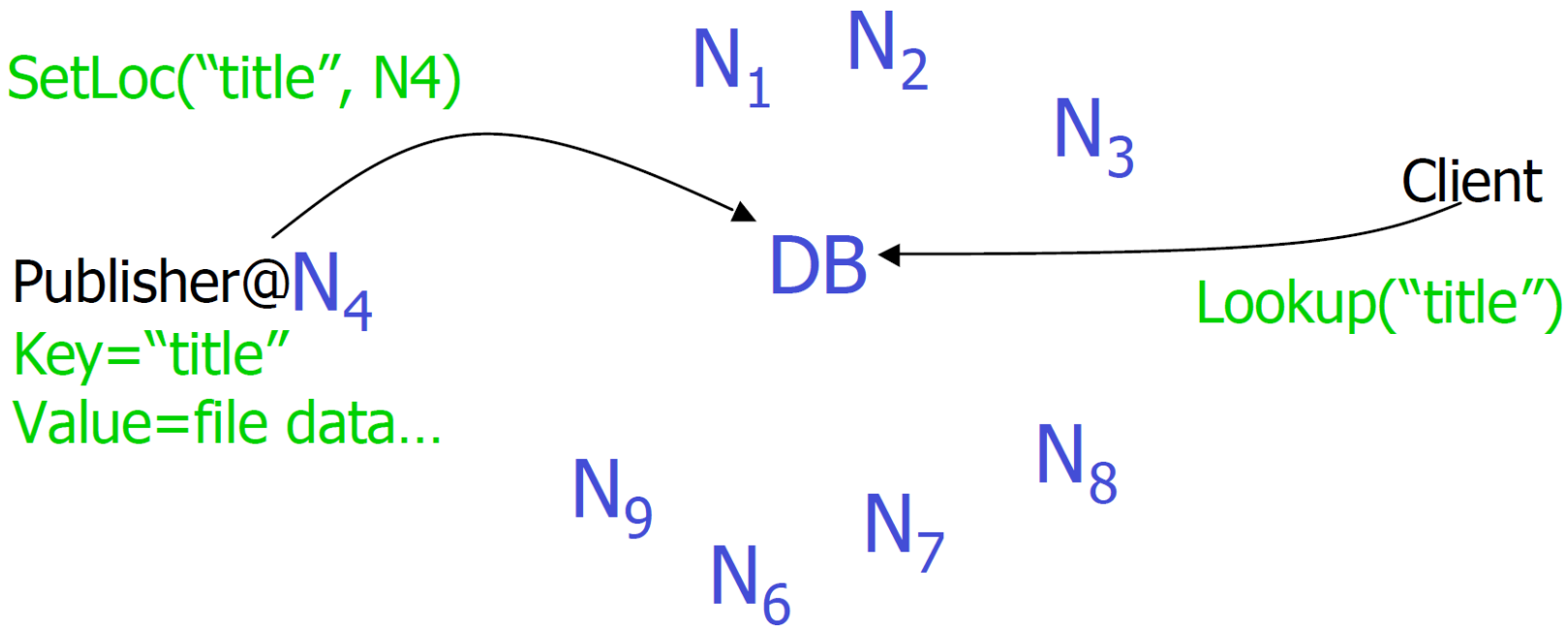
Obrázek: *Hybridní P2P systémy:* Požadavek na vyhledání určité informace je nejprve směrován na tzv. *superpeer* (*ultrapeer*) uzel nadřazený dotazujícímu se uzlu; daný uzel ve spolupráci s ostatními superpeer uzly vyhledá uzel, který požadovaná data spravuje, a dotazující se uzel na něj přesměruje.

Směrování v P2P sítích

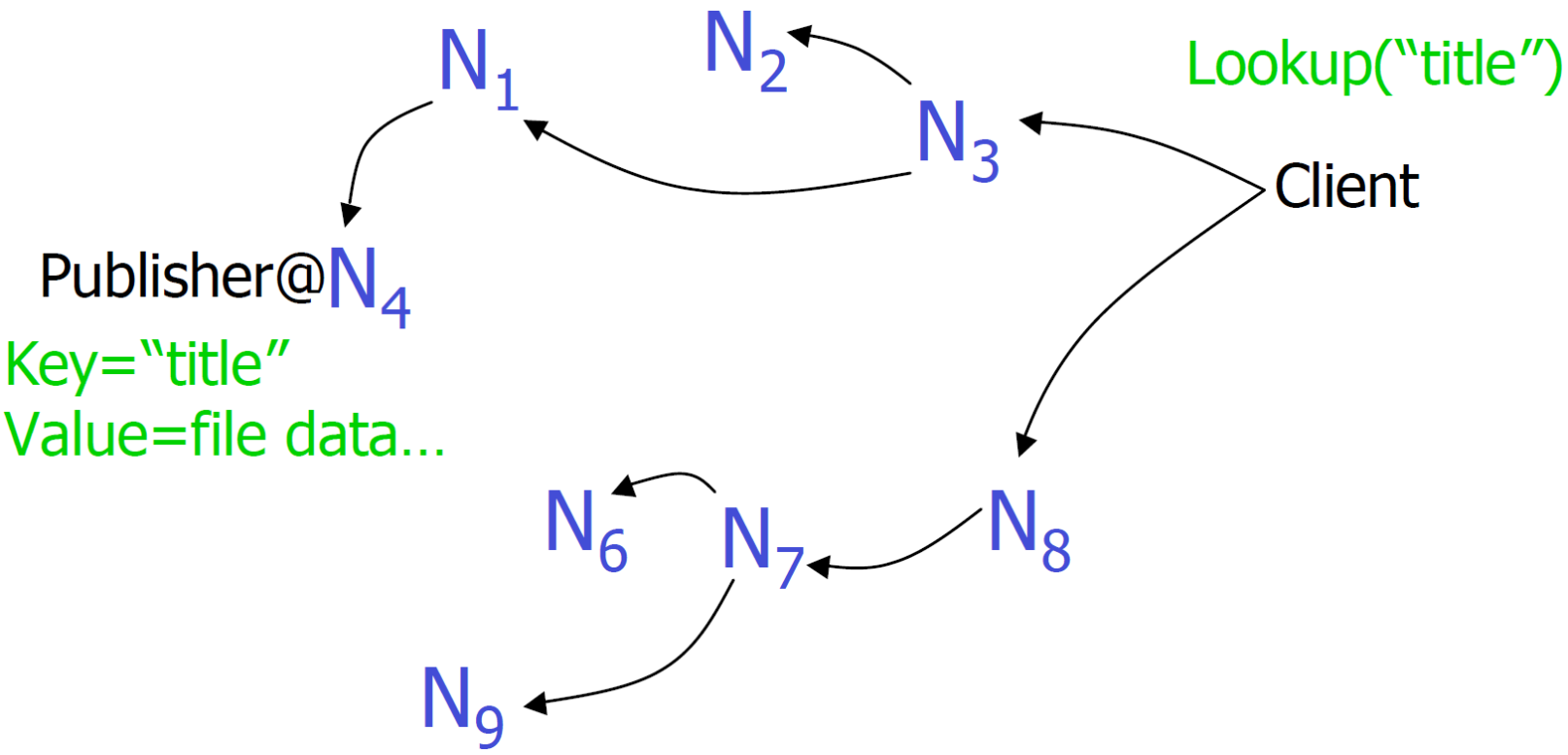
- směrování zpráv/požadavků je jednou z klíčových operací P2P systému
  - pro nalezení požadovaných zdrojů by měl být každý peer schopen přeposlat daný požadavek svému sousedovi, který je blíže cílovému uzlu
  - $\rightarrow$ návrh vhodných směrovacích protokolů je jedním z nejvíce zkoumaných témat v oblasti P2P systémů
- klíčovou vlastností představených směrovacích schémat je množství dat (resp. metadat) spravovaných jednotlivými uzly P2P sítě
  - + způsob organizace daných informací
  - žádná metadata $\Rightarrow$ neexistuje žádná jiná cesta pro vyhledání požadované informace než záplava/broadcast požadavku skrze celou síť



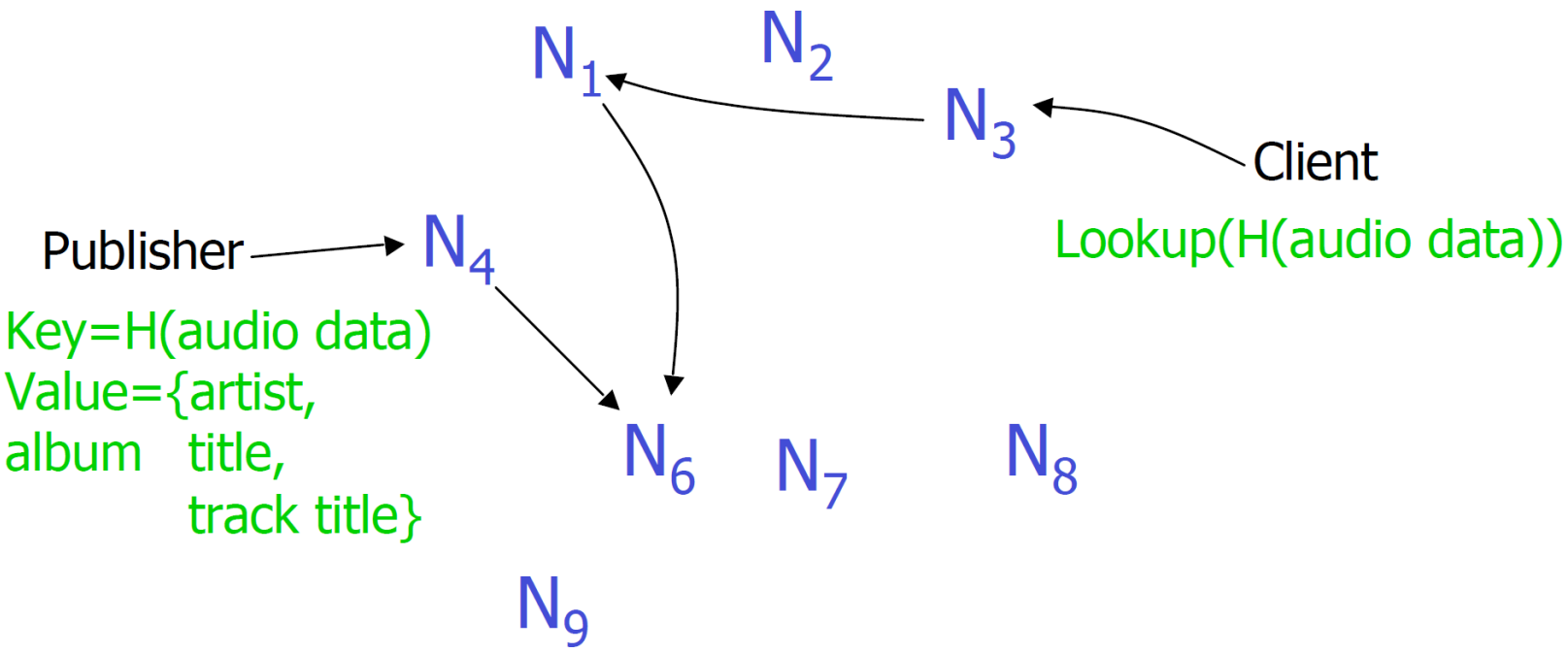

N1
N2
N3
Internet
?
Put (Key="title"
Value=file data...)
Publisher
Client
Get(key="title")
N4
N5
N6



Jednoduché, avšak na jediném (centrálním) uzlu musí být spravováno $O(n)$ stavových informací; centrální uzel je navíc tzv. *single point of failure*.



Robustní, avšak v nejhorším případě je pro každý dotaz sítí přenášeno $O(n)$ zpráv (neškáluje).


N1
N2
N3
Client
Lookup(H(audio data))
Publisher
N4
Key=H(audio data)
Value={artist,
album title,
track title}
N6
N7
N8
N9

Směrování v nestrukturovaných P2P sítích

- každý peer typicky spravuje své vlastní (uložené) datové objekty a linky/spojení ke svým sousedům
- nový uzel pouze kontaktuje nějaký již připojený uzel a zkopíruje si linky k jeho sousedům, čímž si ustaví množinu svých sousedů
  - (která je následně spravována nezávisle na původně kontaktovaném uzlu)
- $\Rightarrow$ žádný z peerů nemá globální znalost o umístění datových objektů v síti
  - vyhledávání probíhá formou záplavy (*flooding*, *broadcast*)
  - pro zmírnění problému záplavy sítě vyhledávacími dotazy je ke každému dotazu připojena hodnota *TTL (Time-To-Live)*, po jejímž vypršení je dotaz zahozen
  - výzvou pro vyhledávací strategie pak je, *jak optimalizovat zpracování dotazu v rámci omezeného množství vyhledávacích kroků (omezeno hodnotou TTL)*
- navrženo mnoho technik pro vyhledávání v nestrukturovaných P2P sítích:
  - *Prohledávání do šířky (Breadth-First Search, BFS)* – e.g., Gnutella
  - *Prohledávání do hloubky (Depth-First Search, DFS)* – e.g., FreeNet
  - *Heuristické směrovací strategie*

Heuristické směrovací strategie
Postupné vnořování (Iterative Deepening)

- **idea:**
  - každý dotaz je prováděn sekvencí několika tradičních BFS prohledávání s postupným zvětšováním prohledávacího dosahu
  - proces vyhledávání je ukončen buď nalezením výsledku nebo dosažením maximální možné hloubky prohledávání
- **detaily algoritmu:**
  - specifikací systémové politiky P je určeno množství a hloubka jednotlivých vyhledávacích iterací
    - $P = D_1, D_2, \ldots, D_n$, kde $D_1 < D_2 < \ldots < D_n$
  - na základě této politiky vysílá zdrojový uzel nejprve vyhledávací dotaz (s využitím techniky BFS) s hloubkou hledání D_1
  - jakmile je nalezen odpovídající výsledek, je vyhledávání ukončeno
  - jinak zdrojový uzel vysílá nový vyhledávácí dotaz (se stejným identifikátorem), nyní však s hloubkou BFS prohledávání D_2
    - uzly, které se nachází blíže než D_1-skoků od zdrojového uzlu tento dotaz pouze přeposílají svým sousedům (již jej nezpracovávají)
    - dále vzdálené uzly dotaz zpracovávají stejně, jako byl zpracováván v první iteraci
  - podobně pro D_3, D_4, atd.
  - pokud není dotaz zodpovězen do hloubky D_n, je prohledávání ukončeno s negativním výsledkem hledání

- **idea:**
  - při tradičním BFS přeposílá každý z uzlů daný dotaz všem svým sousedům
  - při *Řízeném BFS (Directed BFS)* posílá každý z uzlů dotaz pouze podmnožině svých sousedů
  - klíčovou vlastností daného směrovacího algoritmu je inteligentní výběr „vhodných“ sousedů, u kterých existuje předpoklad možného zodpovězení daného dotazu
- **detaily k výběru sousedů:**
  - každý z uzlů si spravuje určité statistiky o svých sousedech: počet dříve zodpovězených dotazů skrze daného souseda, počet získaných výsledků, či zpoždění přijetí výsledků
  - tyto statistiky může uzel využít pro „inteligentní“ výběr sousedů, kterým nový dotaz přepošle, a to na základě mnoha heuristik, například:
    - vybrat souseda, který v minulosti poskytl největší množství výsledků
    - vybrat souseda, který v minulosti odpovídal na dotazy skrze nejmenší počet kontaktovaných uzlů („hop count“)
    - vybrat souseda, který v minulosti přeposlal největší množství zpráv
    - vybrat souseda, který má nejkratší frontu nezpracovaných zpráv
    - atd.

- *Řízené BFS (Directed BFS)*
  - *výhoda:* radikálně menší počet vyhledávacích zpráv oproti standardnímu BFS
  - *nevýhoda:* statistiky vedené vůči každému ze sousedů jsou příliš jednoduché
    - neobsahují žádné informace související s obsahem dotazů
- $\Rightarrow$ **Inteligentní prohledávání (Intelligent Search)**
  - každý peer hodnotí své sousedy na základě jejich relevantnosti k danému dotazu
  - dotaz je pak přeposlán pouze těm sousedům, kteří jsou danému dotazu nejvíce relevantní
  - tímto je dosaženo preciznějšího hodnocení sousedů než u Řízeného BFS
    - dobrých výsledků je dosaženo zejména na sítích s vysokým stupněm lokálnosti dotazů

Heuristické směrovací strategie
Lokální indexy (Local Indices Search)

- **idea:**
  - každý uzel si vytváří a spravuje indexy jak pro svá lokální data, tak pro data svých sousedů, kteří jsou od něj vzdáleni k hopů
    - pokud $k = 0$, je tato metoda podobná tradičnímu BFS (uzly spravují indexy pouze lokálních dat)
  - výsledek navrácený takovýmto uzlem je pak stejný jako výsledek, který by byl navrácen všemi uzly v rozsahu k hopů od daného uzlu
- **detaily:**
  - dotazy jsou zpracovávány podle globální politiky P, která specifikuje hloubky, ve kterých jsou dotazy zpracovávány
    - dotaz zpracovávají pouze uzly v uvedených hloubkách (P) od dotazujícího se uzlu
    - ostatní uzly daný dotaz pouze přeposílají svým sousedům (nezpracovávají jej)
- *výhoda:*
  - redukce výpočetní náročnosti díky omezení zpracování dotazu na menší počet uzlů
- *nevýhody:*
  - vyšší požadavky na úložnou kapacitu (větší indexy na uzlech)
  - vysoká režie na údržbu (aktualizaci) těchto indexů
  - nekonzistence/zastaralost indexů (díky dynamice sítě)

- **idea:**
  - jakmile uzel přijme nový dotaz, je tento přeposlán náhodně vybranému sousedovi ke zpracování
  - tento proces se opakuje tak dlouho, dokud není nalezena uspokojivá odpověď
    - nebo dokud neexpiruje TTL (pokud je využito) $\Rightarrow$ výsledek nebyl nalezen
- **detaily:**
  - *hlavní nevýhoda:*
    - trpí na dlouhé latence zpracování dotazů
  - $\Rightarrow$ *k-walker Random Walk Algorithm*
    - původce dotazu (zdrojový uzel) zasílá k dotazovacích zpráv svým náhodně vybraným sousedům (místo jedné jediné zprávy = originální 1-*walker algoritmus*)
    - jakmile uzel přijme dotazovací zprávu (tzv. *walker*), zpracovává/přeposílá ji s využitím stejného principu jako v případě základního algoritmu náhodné procházky (tj. *jedinému* vybranému sousedovi)
    - počet zpráv (navštívených uzlů) je navýšen *lineárně* v porovnání s 1-walker algoritmem

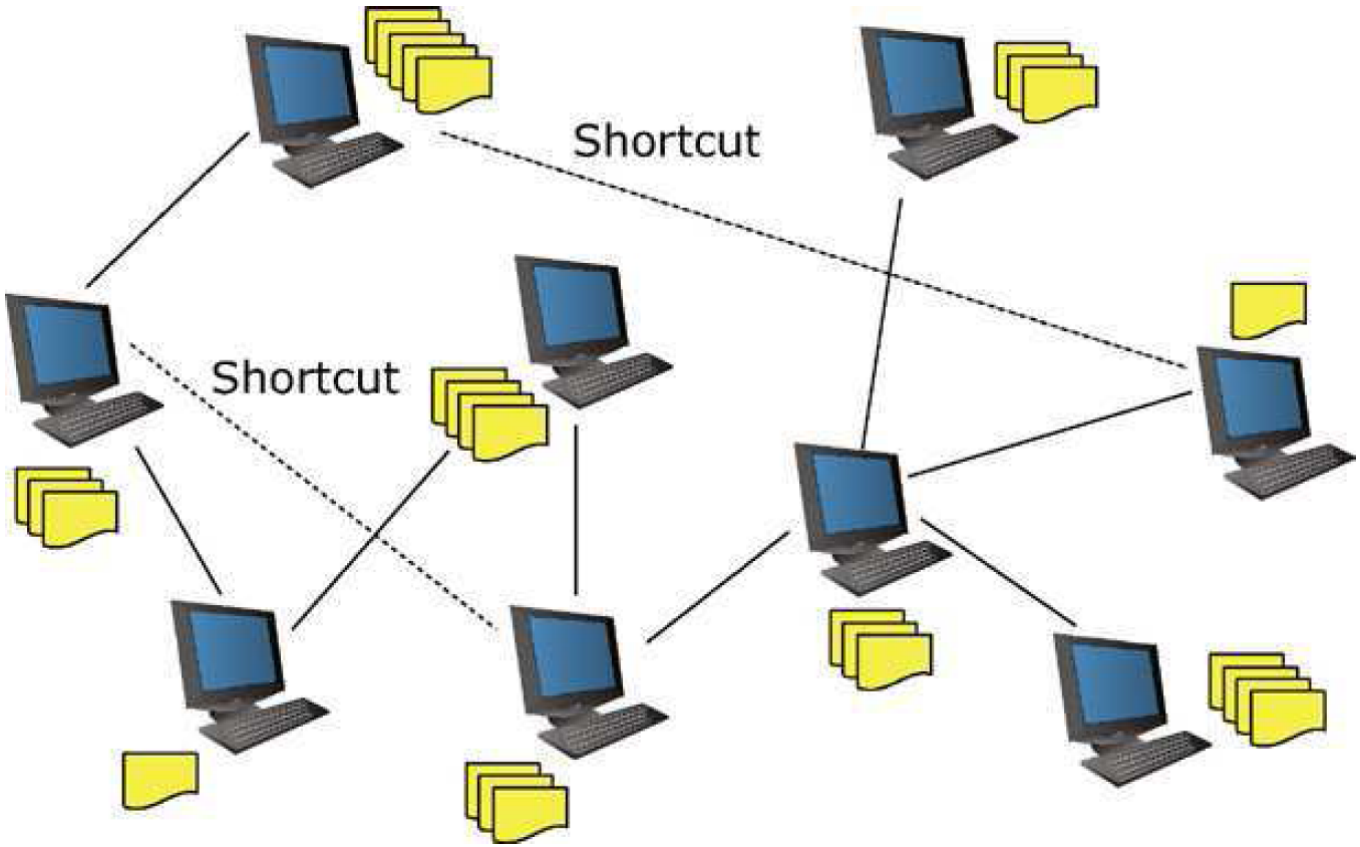
- **detaily (pokračování):**
  - $\Rightarrow$ *Random Breadth First Search (RBFS)*
    - podobné k-walker Random Walk
    - původce dotazu nejprve náhodně vybere podmnožinu svých sousedů, kterým dotaz přepošle
    - každý z těchto sousedů pak opět náhodně vybírá *podmnožinu* svých sousedů, kterým dotaz přepošle
    - atd.
    - počet zpráv (navštívených uzlů) je navýšen *exponenciálně* v porovnání s 1-walker algoritmem

- **idea:**
  - prohledávací metoda, která kombinuje k-walker náhodnostní prohledávání a pravděpodobnostní prohledávání
  - hlavní rozdíl mezi APS a algoritmy na bázi náhodnostních „walkers“:
    - náhodností walkers zasílají dotaz náhodně vybraným sousedům, zatímco APS zasílá dotaz sousedům na vybraným na základě určité pravděpodobnosti
    - $\Rightarrow$ na základě dřívějších výsledků spravuje každý uzel pro každého ze svých sousedů pravděpodobnost jeho vybrání (pro každý objekt, resp. kategorii objektů)
- **detaily:**
  - dva přístupy aktualizace spravovaných pravděpodobností:
    - *Optimistický přístup* – systém proaktivně navyšuje pravděpodobnost aktuálně vybraným (=dotazovaným) sousedům a snižuje jejich pravděpodobnost pouze tehdy, když „walker“ přes ně vyslaný skončí s chybou
    - *Pesimistický přístup* – systém proaktivně snižuje pravděpodobnost aktuálně vybraným (=dotazovaným) sousedům a zvyšuje jejich pravděpodobnost pouze tehdy, když „walker“ přes ně vyslaný skončí úspěšným nalezením výsledku

- **detaily (pokračování):**
  - *swapping-APS* – každý uzel průběžně přepíná mezi optimistickým a pesimistickým přístupem
    - na základě pozorování poměru uspěšných a neúspěšných „walkerů“ (pozorováno pro každý objekt)
  - *weighted-APS* – bere v úvahu také lokalitu sousedů (bližší uzly mají vyšší pravděpodobnost)

- **idea:**
  - každý uzel si spravuje doplňkové linky pro účely vyšší efektivity prohledávání
    - tyto linky (nazývané *interest-based short-cuts*) spojují dva uzly mající podobný „zájem“ (podobná data)
- **detaily:**
  - při zpracování dotazu uzlem jsou nejprve konzultovány tyto doplňkové linky (dotaz je přeposlán daným uzlům)
    - pokud je výsledek nalezen, prohledávání končí
    - jinak je použita běžná prohledávací metoda
  - *vytváření zkratek (shortcutů):*
    - při připojení uzlu do systému nemá tento žádné shortcuts
    - po každém úspěšně zodpovězeném dotazu si původce dotazu do své databáze přidá doplňkové linky k uzlům, ktere na dotaz odpověděly
    - každý z uzlů si ukládá/spravuje pouze omezený počet shortcutů (kvůli limitům na úložné kapacity)

Heuristické směrovací strategie
Interest-Based Shortcuts II.



Směrování ve strukturovaných sítích I.

- nestrukturované P2P sítě trpí problémem nedostatečné efektivity vyhledávání
- narozdíl od nestrukturovaných sítí, uzly strukturované P2P sítě jsou organizovány do určité fixní topologie (uspořádání)
  - například kruh (Chord), vícerozměrná mřížka (CAN), mesh (Pastry and Tapestry) či několika seznamů (Skip Graph)
  - $\Rightarrow$ nově připojený uzel se musí řídit definovanými procedurami za účelem zjištění a začlenění se na definovanou pozici v síti
- lze garantovat, že pokud výsledek na zadaný dotaz v síti existuje, bude vždy nalezen
  - navíc efektivně – většina systémů dokáže na dotazy zodpovídat se složitostí $O(log\ N)$ kroků/zpráv (N = počet uzlů v síti)
- *nevýhoda:*
  - nutnost udržování definované topologie vyžaduje další režii v podobě správy/udržování směrovacích tabulek
- podle využité překryvné topologie lze strukturované P2P systémy dělit do následujících kategorií:
  - *Systémy využívající distribuované hashovací tabulky* – např., Chord, CAN, Tapestry a Pastry, Viceroy a Crescendo, atd.
  - *Systémy využívající tzv. Skip-listy* – např., Skip Graph, SkipNet, atd.
  - *Systémy využívající stromové struktury* – např., P-Grid, P-Tree, BATON, atd.

Směrování ve strukturovaných sítích II.

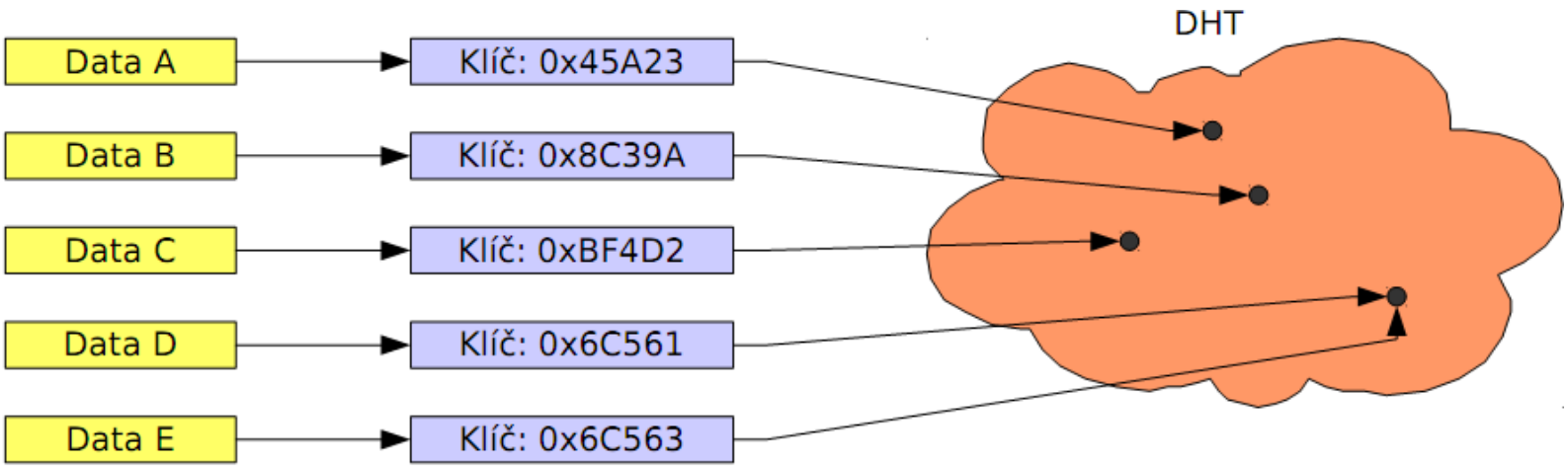
- *Distributed hash tables, DHT* (distribuované hash tabulky)
  - Řešení pro účinné hledání i řídce se vyskytujících objektů
- Hash tabulky
  - Rychlé a levné nalezení dat indexovaných podle klíče
  - Výše jsme neomezovali způsob vyhledání dat
  - Pokud použijeme hash tabulky, pak ztrátu obecnosti musíme nějak kompenzovat (ale často i vyhledání podle klíče – např. název objektu – plně stačí)
- Obětujeme obecnost za efektivitu a rychlost

Distribuované hash tabulky (DHT)

- Hash tabulky v p2p sítích: hledáme data podle klíče
- Předpokládáme, že data jsou uložena v distribuovaných uzlech (ne v poli)
- Důsledky:
  - Použijeme překryvovou síť která spojuje uzly pro nás vhodným způsobem
  - Počet uzlů neznáme a navíc se mění v čase
  - Pracujeme s idealizovanými vlastnostmi
    - Hash funkce mapuje na idealizovaný prostor
    - Samotný prostor je mapován na uzly dynamicky

Distribuované hash tabulky (DHT)

- každý uzel P2P sítě spravuje určitou část globální hashovací tabulky
- uložení/zjištění položky s znamená oslovení uzlu, který spravuje hodnotu $hash(s)$



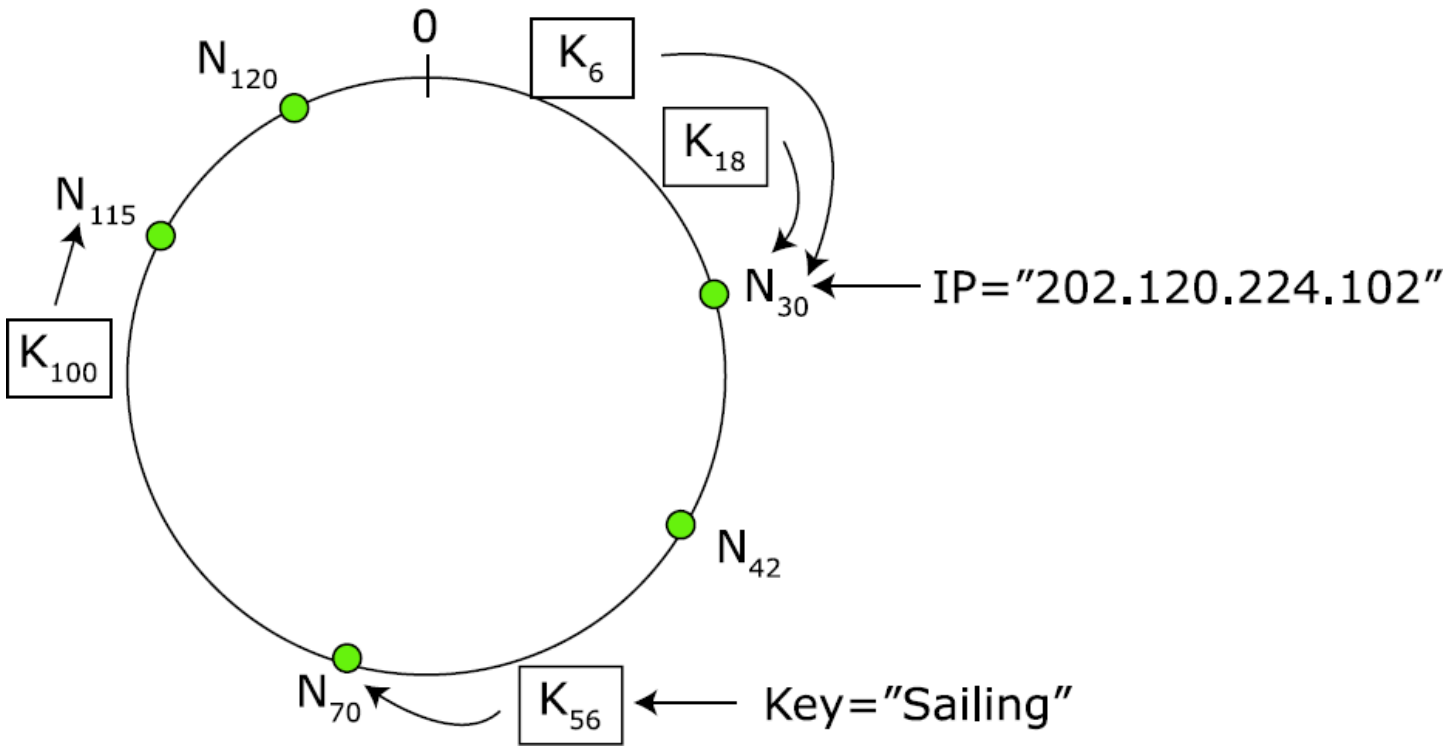
Hlavní principy

- Přiřadíme klíči jedinečný uzel
- Nalezneme tento uzel rychle a snadno v překryvové síti
- Zavedeme replikaci (více uzlů pro jeden objekt/klíč)
- Rozložení zátěže může dynamicky měnit přiřazení klíče

Nejrozšířenější jednoduchá implementace: **Chord**

- Velmi populární (více jak 3000 citací).
- Překryvová síť: orientovaná kružnice

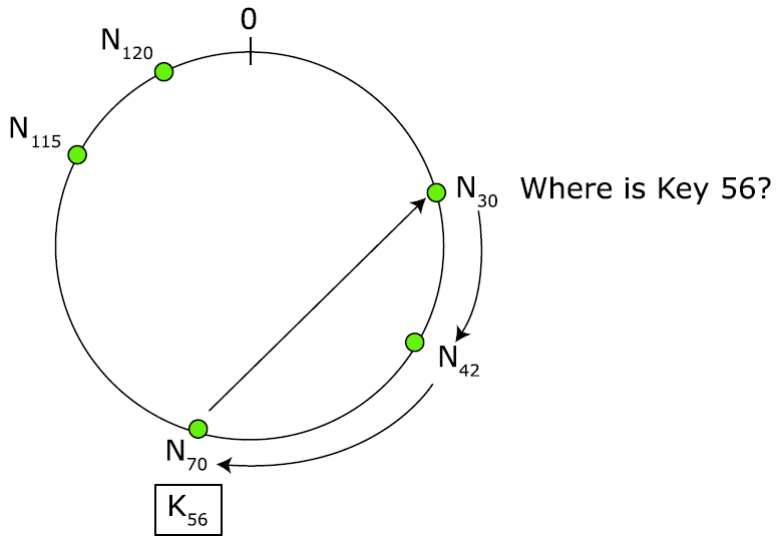
- jeden z nejznámějších mechanismů směrování ve strukturovaných P2P sítích
- **idea:**
  - využívá jednosměrnou hashovací funkci pro mapování jak uzlů, tak datových položek do jednorozměrného prostoru m-bitových identifikátorů
    - pro vygenerování identifikátoru uzlu je využita IP adresa uzlu, a
    - pro vygenerování identifikátoru datové položky je využita vlastní položka (případně jiný klíč, metadata, atp.)
    - prostor identifikátorů je zapotřebí zvolit dostatečně velký, aby pravděpodobnost přiřazení stejného identifikátoru dvěma různým uzlům sítě byla minimální
- **detaily:**
  - prostor identifikátorů je logická kružnice čísel v rozsahu $[0, 2^m - 1]$
  - systém přiřazuje datovou položku s klíčem k prvnímu uzlu n, jehož identifikátor je roven, případně je vyšší než identifikátor k (v prostoru kružnice)
    - t.j., datová položka s klíčem k je přiřazena prvnímu uzlu ve směru hodinových ručiček od k



Obrázek: Kružnice identifikátorů založená na logické kružnice – klíče K_6 a K_{18} jsou přiřazeny stejnému uzlu s identifikátorem N_{30} (zjištěn hashováním jeho IP adresy „202.120.224.102“). Klíč K_{56} (získán hashováním slova „Sailing“) je přiřazen uzlu s identifikátorem N_{70}; klíč K_{100} je přiřazen uzlu s identifikátorem N_{115}; uzly N_{42} a N_{120} nemají žádné datové položky.

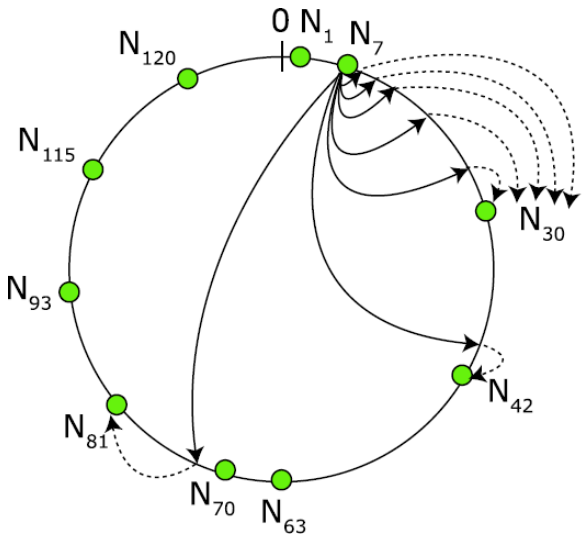
Jednoduchý vyhledávací algoritmus:

- každý uzel zná pouze svého *bezprostředního následovníka*
- jakmile uzel přijme požadavek na vyhledání:
  - nejprve ověří, zda on sám nemá požadovanou datovou položku
    - pokud ano, je výsledek dotazu navrácen dotazujícímu uzlu
  - pokud ne, přeposílá požadavek svému bezprostřednímu následovníkovi
  - vyhledávání končí, jakmile
    - je výsledek dotazu nalezen
    - je identifikátor bezprostředního následovníka větší než identifikátor vyhledávané datové položky $\Rightarrow$ výsledek dotazu nelze nalézt
  - složitost vyhledávání je $O(N)$ (N = počet uzlů v systému)



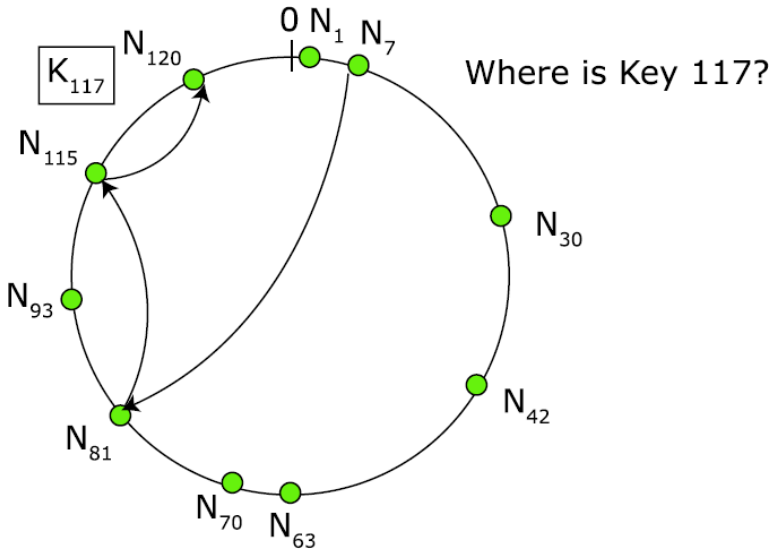
Škálovatelný vyhledávací algoritmus:

- místo spravování pouze svého bezprostředního následovníka spravuje každý uzel svou vlastní směrovací tabulku (tzv. *finger table*) obsahující záznamy k jeho x následovníkům
- jakmile uzel n přijme požadavek na vyhledání datové položky s identifikátorem k:
  - jestliže daný uzel nespravuje vyhledávaná data, prohledává svou směrovací tabulku a hledá takový uzel n, pro který platí podmínka $n.id < n'.id < k$
    - pokud takový uzel existuje, přepošle mu uzel n požadavek na vyhledání hledané datové položky
    - jinak uzel žádá svého bezprostředního následovníka, aby hledanou datovou položku vyhledal
  - vyhledávání končí, jakmile
    - je výsledek dotazu nalezen
    - je identifikátor bezprostředního následovníka větší než identifikátor vyhledávané datové položky $\Rightarrow$ výsledek dotazu nelze nalézt
  - složitost vyhledávání je $O(log\ N)$ (N = počet uzlů v systému)



Finger Table

N_{7+1}	N_{30}
N_{7+2}	N_{30}
N_{7+4}	N_{30}
N_{7+8}	N_{30}
N_{7+16}	N_{30}
N_{7+32}	N_{42}
N_{7+64}	N_{81}



Obrázek: Příklad směrovací tabulky (vlevo) a příklad vyhledání datové položky s identifikátorem K_{117} začínající v uzlu N_7 (vpravo).

Algorithm 2 : *Chord_Lookup* (Node n, Key-Id k)

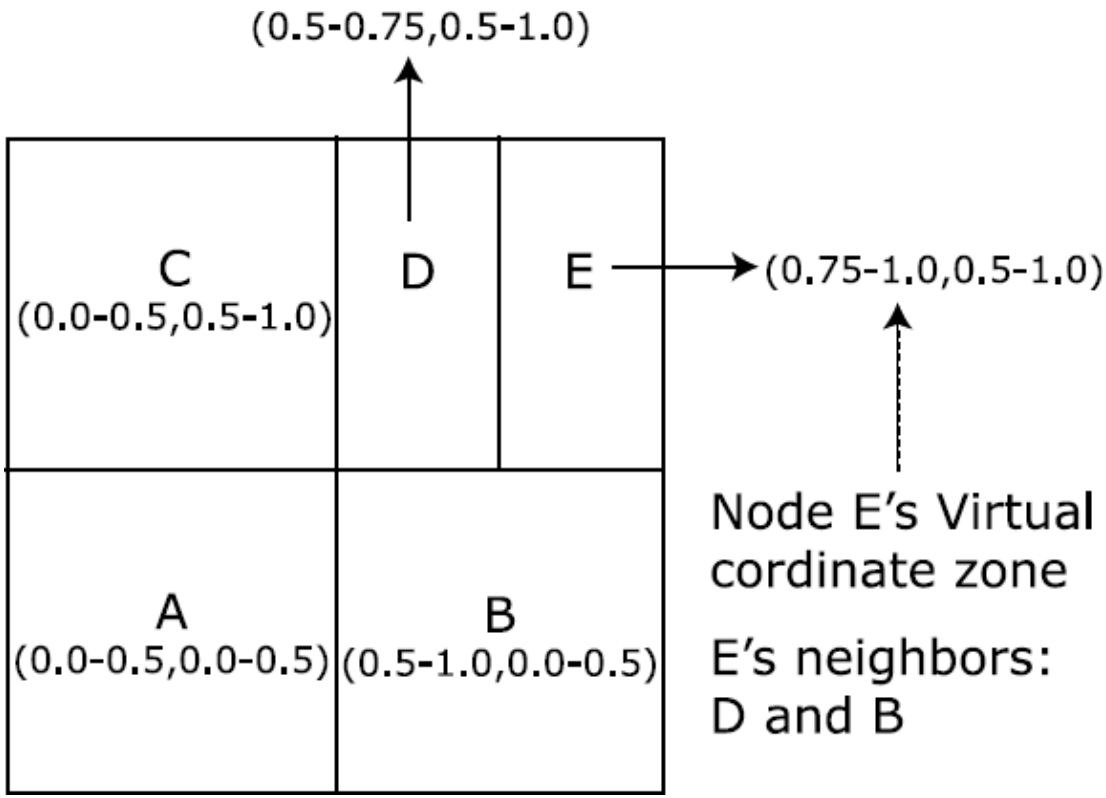
```
1:  for i = m down to 1 do
2:      n' = n.routingtable.get(i)
3:      if n.id < n'.id < k then
4:          Chord_Lookup(n', k)
5:          return
6:      end if
7:  end for
8:  n' = n.successor
9:  if n.id < n'.id < k then
10:     Chord_Lookup(n', k)
11: else
12:     result = Local_Search(k)
13:     return result to the query issuer node
14: end if
```

Budování systému:

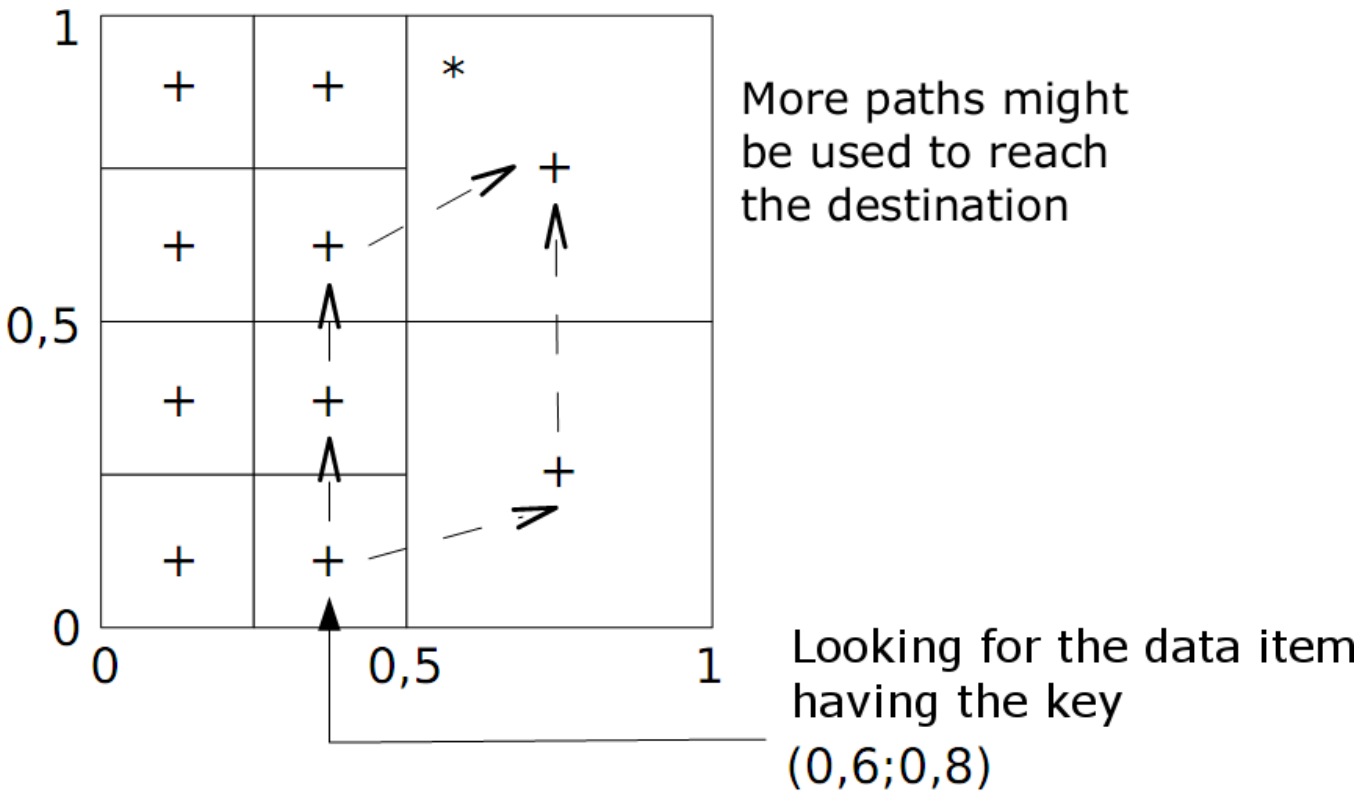
- při připojení nového uzlu tento:
  1. hledá svou pozici v Chord kruhu a přejímá data, za které by měl být zodpovědný (na základě klíčů)
  2. inicializuje své směrovací tabulky
  3. aktualizuje směrovací tabulky ostatních uzlů, které by měly reflektovat danou změnu
- při odpojení existujícího uzlu není potřeba žádných kroků

Směrovací strategie založené na DHT
Content Addressable Network (CAN) I.

- **idea:**
  - směrovací systém utvořený nad virtuálním d-dimenzionálním kartézským souřadnicovým systémem
    - systém rozděluje prostor datových klíčů do samostatných oblastí, kdy každá z oblastí je spravována jediným (vybraným) uzlem
    - tento uzel spravuje všechna datové položky spadající do jím spravované oblasti
    - systém využívá hashovací funkce pro mapování datové položky na bod p v souřadnicovém systému (klíčem je tak d-tice)
- **detaily:**
  - vkládání nové datové položky:
    1. hodnota datové položky je namapována na bod p souřadnicového systému
    2. je nalezen uzel n, do jehož zóny spadá p – tento je kontaktován a požádán o uložení položky
  - zpracování vyhledávacího dotazu je podobné
    - pokud výsledek existuje, musí být uložen na uzlu pokrývajícím danou zónu
  - každý z uzlů musí spravovat informace o svých sousedech
    - to jest, uzly spravující přilehlé zóny
  - směrování je založeno na jednoduchém hladovém algoritmu
    - v každém kroku je hledán uzel, jehož souřadnice jsou blíže cílové zóně



Obrázek: CAN P2P systém využívající dvoudimenzionální prostor spravovaný 5 uzly.



Obrázek: Příklad vyhledání datové položky v CAN P2P systému.

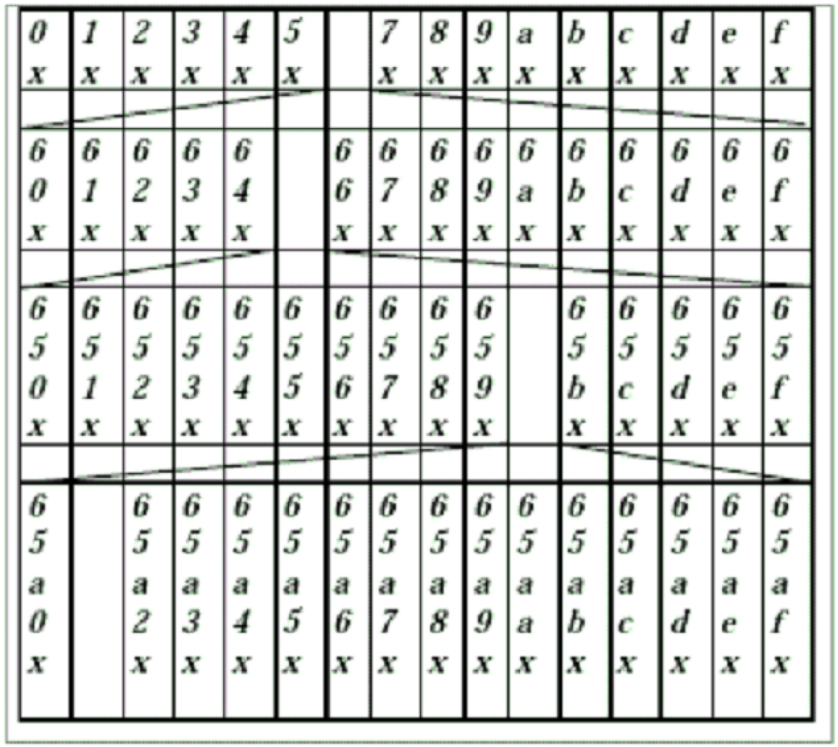
Algorithm 3 : *CAN_Lookup* (Node n, Point p)

```
1: if n.zone covers p then
2:     result = Local_Search(p)
3:     return result to the query issuer node
4: else
5:     find a neighbor node n' whose zone is closer to p
6:     CAN_Lookup(n', p)
7: end if
```

Budování systému:

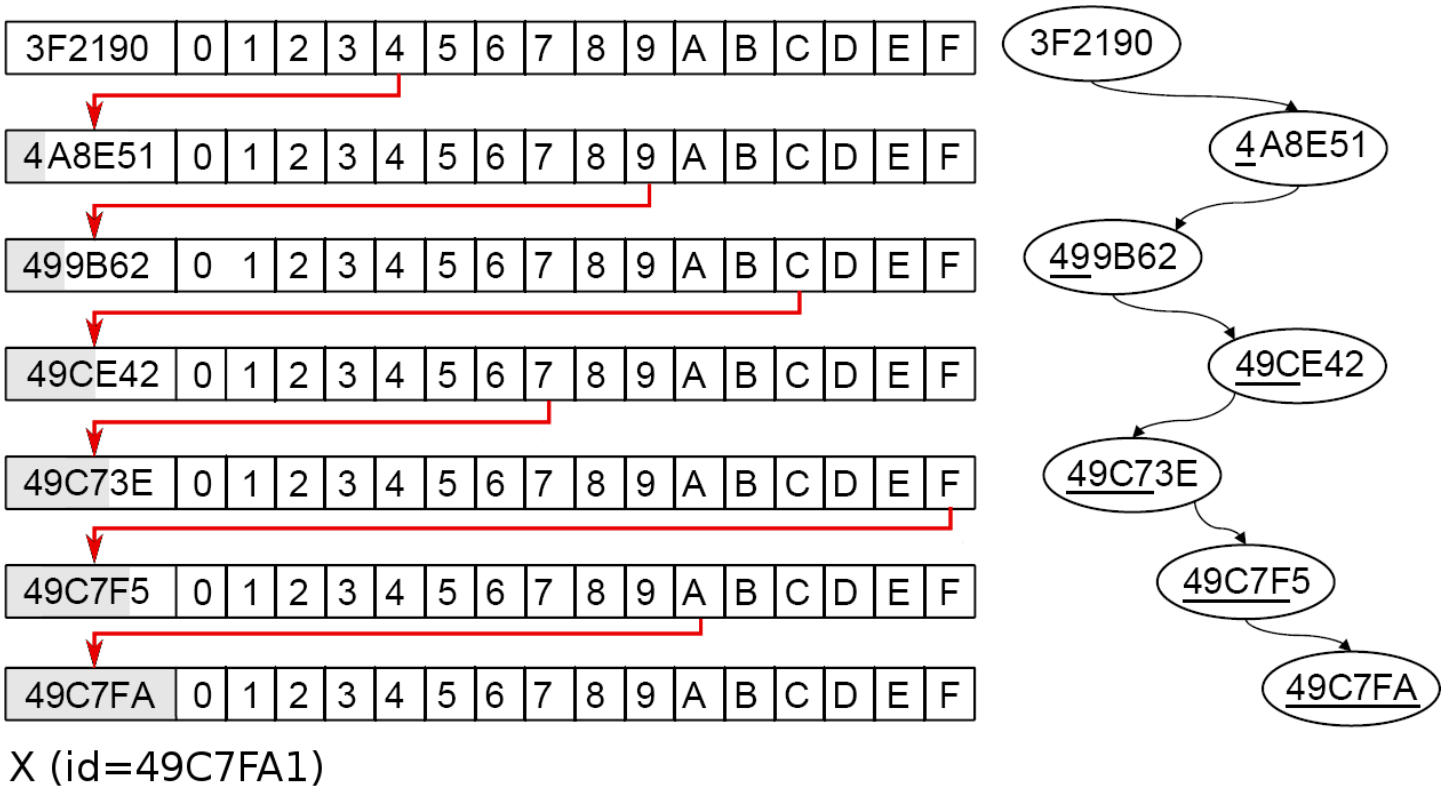
- při připojení nového uzlu k systému musí tento:
  1. najít libovolný, již připojený uzel
  2. identifikovat zónu, která může být rozdělena, a požádat jejího vlastníka/správce o její rozdělení na dvě části
     - jedna část bude spravována původním uzlem, druhá část bude spravována nově připojeným uzlem
  3. vytvořit si svou směrovací tabulku a zaktualizovat směrovací tabulky svých sousedů
- při odpojení uzlu ze systému musí tento požádat vybraného souseda o sloučení obou zón do jediné

- **idea:**
  - směrovací systém založený na tzv. *PRR stromech*
    - PRR = Plaxton, Rajaraman a Richa (1997)
  - m-bitový identifikátor uzlu je rozdělen na sekvenci „číslic“ se základem 2^b
    - např. 128-bitový identifikátor je rozdělen na 32 4-bitových číslic ($b = 4$, základ $= 2^4 \Rightarrow$ hexadecimální sekvence číslic)
    - b . . . konfigurovatelný parametr
  - daná datová položka je pak uložena na uzel, jehož identifikátor sdílí *nejdelší společný prefix s identifikátorem dané datové položky*
  - v každém směrovacím kroku je hledán/vybrán uzel, který má s cílovým uzlem (datovou položkou) delší společný prefix než hledající uzel (delší o 1 číslici, tj. b bitů)
    - složitost hledání je $O(log_{2^b} N)$
- **detaily:**
  - každý peer má svou *směrovací tabulku*
    - organizovaná do pevného počtu řádků (tzv. levelů) $(= \lceil log_{2^b}(N) \rceil)$ a v rámci každého řádku/levelu pak obsahující pevný počet položek $(= 2^b - 1)$
    - ID řádku = délka společného prefixu s hledaným uzlem
    - ID sloupce = další možný krok



0	1	2	3	4	5		7	8	9	a	b	c	d	e	f
x	x	x	x	x	x		x	x	x	x	x	x	x	x	x
6	6	6	6	6		6	6	6	6	6	6	6	6	6	6
0	1	2	3	4		6	7	8	9	a	b	c	d	e	f
x	x	x	x	x		x	x	x	x	x	x	x	x	x	x
6	6	6	6	6	6	6	6	6	6		6	6	6	6	6
5	5	5	5	5	5	5	5	5	5		5	5	5	5	5
0	1	2	3	4	5	6	7	8	9		b	c	d	e	f
x	x	x	x	x	x	x	x	x	x		x	x	x	x	x
6		6	6	6	6	6	6	6	6	6	6	6	6	6	6
5		5	5	5	5	5	5	5	5	5	5	5	5	5	5
a		a	a	a	a	a	a	a	a	a	a	a	a	a	a
0		2	3	4	5	6	7	8	9	a	b	c	d	e	f
x		x	x	x	x	x	x	x	x	x	x	x	x	x	x

Obrázek: Směrovací tabulka uzlu **65a1**. *x* reprezentuje směrovací informaci (další krok) pro uzel mající daný prefix. (Prázdná místa odpovídají prefixům, které jsou identické s daným uzlem.)



Obrázek: Příklad vkládání/hledání objektu X s identifikátorem $49C7FA1$ do Pastry sítě (počátek hledání/vkládání je na uzlu $3F2190$).

- **detaily pokračování:**
  - kromě směrovací tabulky má každý z uzlů navíc další seznam – tzv. *leaf set*
    - obsahuje seznam ukazatelů na uzly, které jsou k danému uzlu numericky nejblíže
    - slouží jako záložní řešení v okamžiku, kdy ve směrovací tabulce není nalezen uzel s delším prefixem
    - velikost je typicky 2^b – polovina záznamů obsahuje uzly s identifikátorem menším než je identifikátor daného uzlu a druhá polovina pak uzly s identifikátorem větším
- **směrování v Pastry:**
  - pokud je hledaný uzel v leaf set daného uzlu, je směrování provedeno na základě této informace
  - jinak je konzultována směrovací tabulka
    - zpráva je přeposlána uzlu, který s cílovým uzlem sdílí delší společný prefix
  - jestliže je směrovací tabulka prázdná nebo referencovaný uzel je nedostupný, je zpráva přeposlána:
    - sousednímu uzlu majícímu stejně dlouhý prefix, nebo
    - uzlu jehož identifikátor je numericky blíže hledanému klíči
    - (vzácný případ)

Algorithm 4 : *PRR_Routing* (Node j, Node i)

1: find a neighbor node k having the longest matching prefix with i
2: set $p =$ length of matching prefix between i and j
3: set $q =$ length of matching prefix between i and k
4: **if** $q > p$ **then**
5: PRP_Routing(k, i)
6: **else**
7: j as the closest node to i
8: **end if**

Budování systému:

- nově připojený uzel (s identifikátorem X):
  1. kontaktuje libovolný (již připojený) uzel A a zasílá mu zprávu *join(X)*
  2. uzel A směruje zprávu *join(X)* uzlu Z, který je nejblíže klíči X
  3. uzel X přijme leaf set uzlu Z jako svou vlastní a naplní si svou směrovací tabulku (i-tý řádek tabulky je převzat z i-tého uzlu na cestě z A do Z)
  4. uzel X informuje uzly, které by si jej měly přidat do svých směrovacích tabulek

- při odpojení existujícího uzlu ze sítě:
  - musí tento předat jim spravovaná data vybranému sousedovi
  - směrovací tabulky se časem zupdatují automaticky
    - daný uzel bude nahrazen uzlem z jeho nejbližšího okolí (získán z jeho leaf set)

Tapestry:

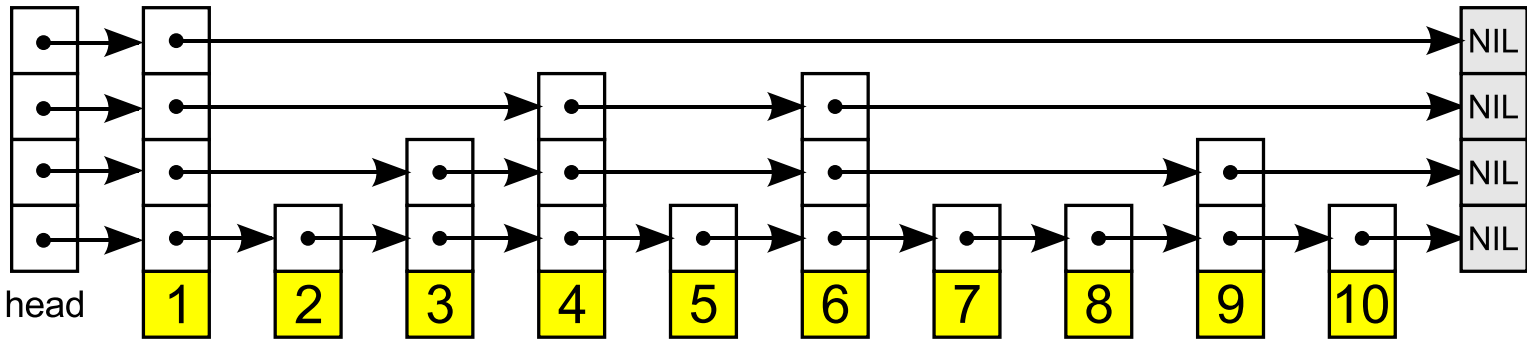
- další směrovací strategie pro P2P sítě založená na PRR stromech
  - velmi podobná Pastry (vznikaly však nezávisle na sobě)
- hlavní rozdíl mezi Pastry a Tapestry:
  - v rámci Pastry se v každém kroku hledání prodlužuje nejdelší společný *prefix* (s hledaným uzlem)
  - v rámci Tapestry se v každém kroku hledání prodlužuje nejdelší společný *suffix* (s hledaným uzlem)
  - (lze identifikovat i několik dalších, většinou drobných rozdílů)

	CAN	Chord	Pastry
Routing performance	$O(d * N^{1/d})$	$O(\log N)$	$O(\log_B N)$
Routing state	$2d$	$\log N$	$B * \log_B N + B$
Peers join/leave	$2d$	$(\log N)^2$	$\log_B N$

$B = 2^b$

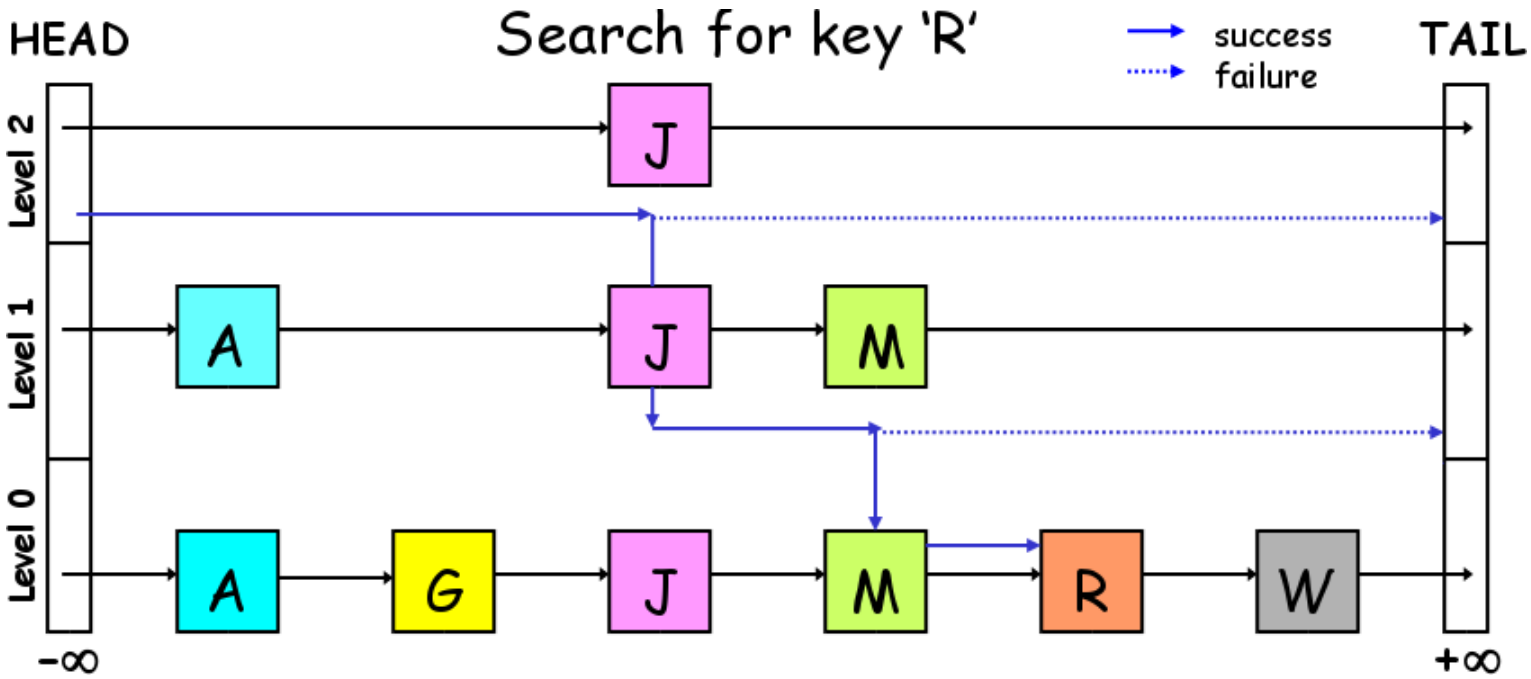
Obrázek: Srovnání prezentovaných směrovacích mechanismů strukturovaných P2P sítí založených na DHT (co do rychlosti/účinnosti hledání, nezbytné úložné kapacity na jednotlivých uzlech a složitosti reorganizace při připojení/odpojení uzlu).

- **skip list** je datová struktura určená pro uložení setříženého seznamu položek, která (pro snadné vyhledání/uložení) využívá hierarchii zřetězených seznamů
- seznamy jsou sestaveny do úrovní:
  - nejnižší úroveň (level 0) je běžný setřízený seznam
  - každá z vyšších vrstev pak funguje jako „expresní linka“ pro seznamy nižších úrovní
    - element z vrsty i se vyskytuje ve vrstvě $i+1$ s určitou pravděpodobností p
    - (obvykle $p = 1/2$ or $p = 1/4$)



Hledání cílového elementu:

- začíná na začátku seznamu umístěného v nejvyšší úrovni a pokračuje horizontálně, dokud není nalezena položka, která je větší nebo rovna hledané
  - pokud je nalezena položka, která je rovna hledané, je cíl nalezen
  - pokud je nalezená položka větší než hledaná, procedura se po navrácení se na předcházející položku a sestoupení o jednu úroveň níže opakuje
- *předpokládaná cena hledání* je $(log_{1/p}\, n)/p$
  - p je konstanta $\Rightarrow O(log\, n)$



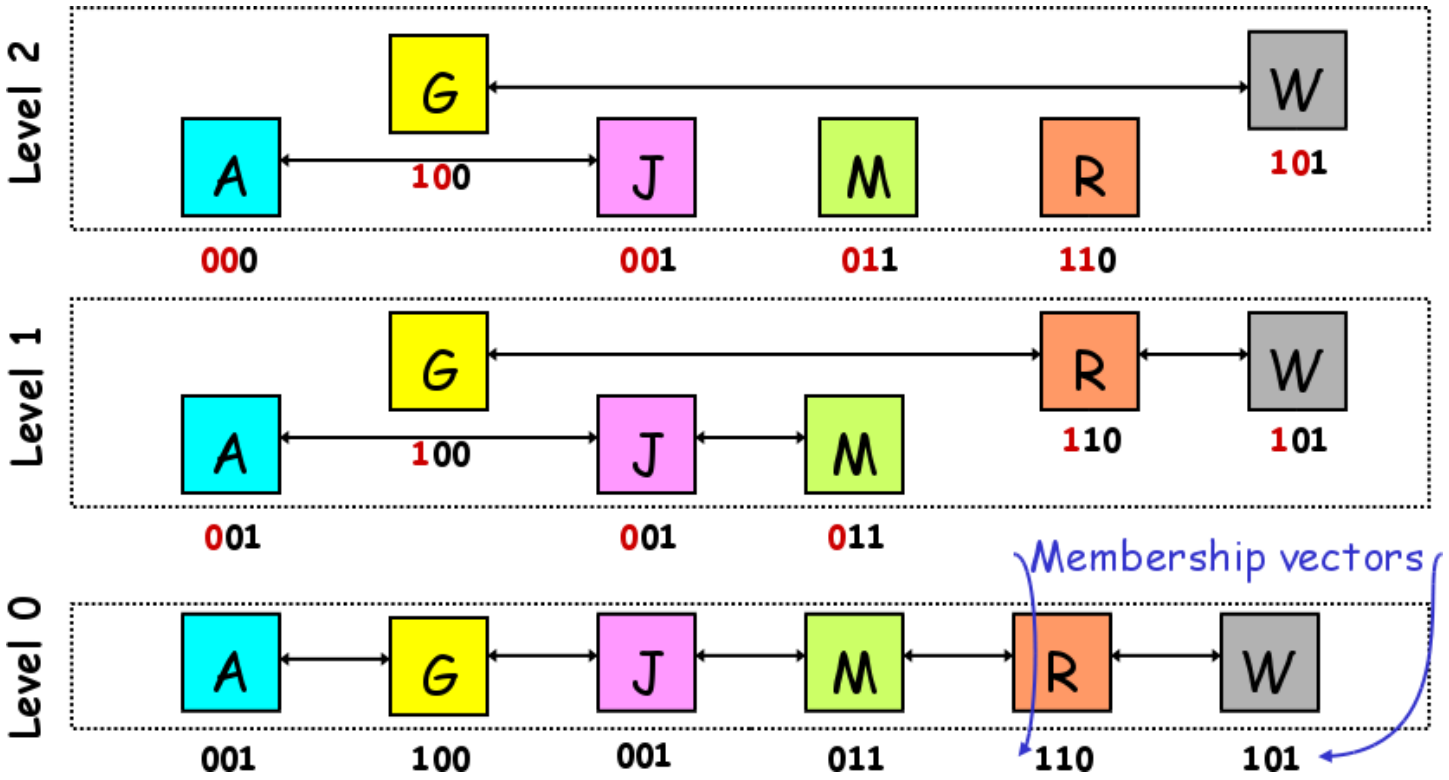
Obrázek: Proces hledání ve Skip List struktuře.

idea:

- směrovací strategie založená na *Skip Listech*
  - originální skip-listy však nejsou vhodné, neboť by docházelo k přetěžování elementů na nejvyšší úrovni
- narozdíl od originálních Skip-listů, kdy tyto využívají na každé úrovni právě jeden seznam, poskytuje Skip Graph *na každé úrovni více takovýchto seznamů*
  - každý uzel je na každé z úrovní umístěn do nějakého seznamu
  - seznamy, do kterých daný uzel náleží, jsou určovány podle náhodného *vektoru příslušnosti (membership vector)* (vytvářen v okamžiku připojování uzlu do systému)
  - počet úrovní je $O(log\ N)$

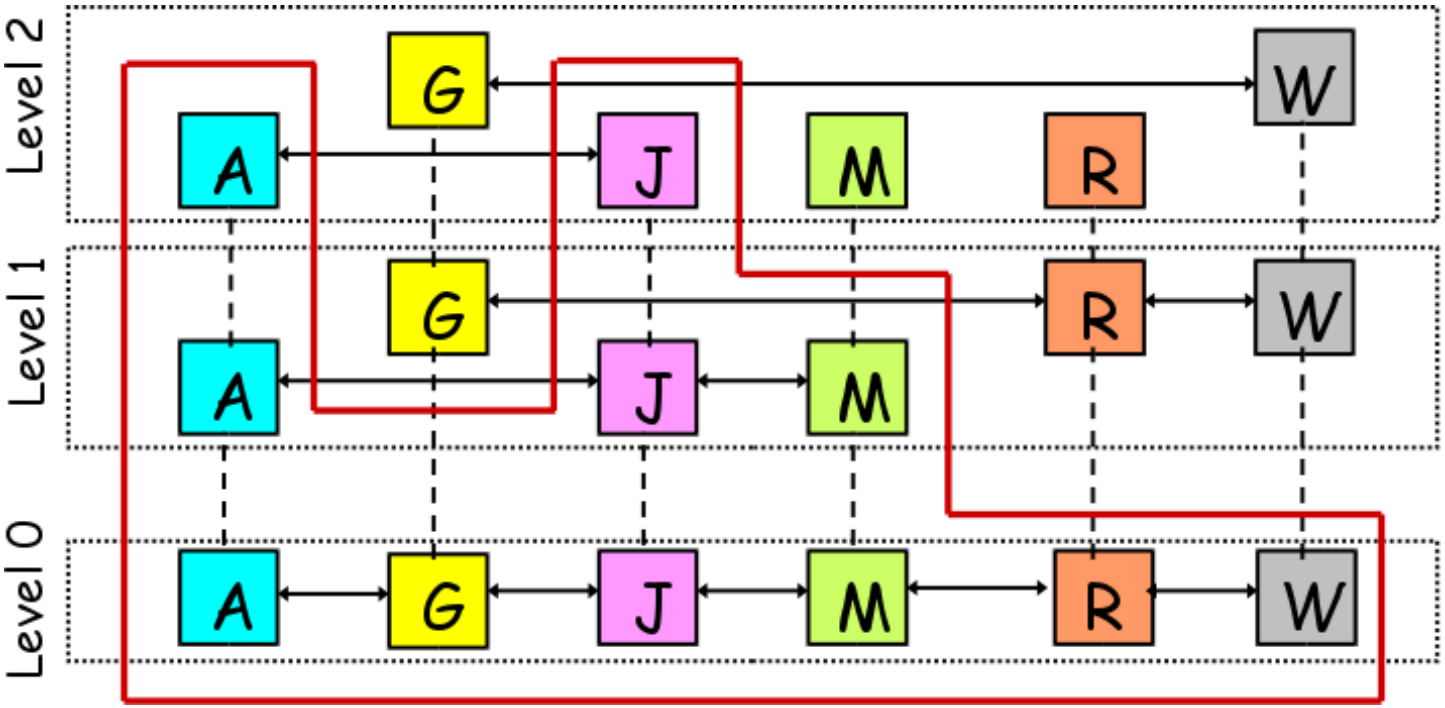
detaily vyhledávání:

- při vyhledávání dotazu uzlem:
  - vyhledávání vždy začíná v nejvyšší úrovni v seznamu, do kterého daný uzel náleží
  - v každém kroku: pokud na dané úrovni existuje uzel, jehož hodnota je blíže vyhledávané, pak je tomuto uzlu dotaz přeposlán
  - jinak daný uzel s vyhledáváním pokračuje na nižší úrovni
  - cílový uzel je nalezen nejhůře v okamžiku sestoupení na nejnižší úroveň (do nejnižšího seznamu)
- složitost vyhledávání je $O(log\ N)$



Vektor příslušnosti (membership vector) definuje pouze seznamy, do kterých daný uzel náleží (*nemá žádnou spojitost s daty, podle kterých jsou seznamy setřízeny*).

Omezením na seznamy obsahující počáteční element/uzel lze získat *skip list* (lze tak využít **stejnou vyhledávací metodu určenou pro běžnou skip-list strukturu**):

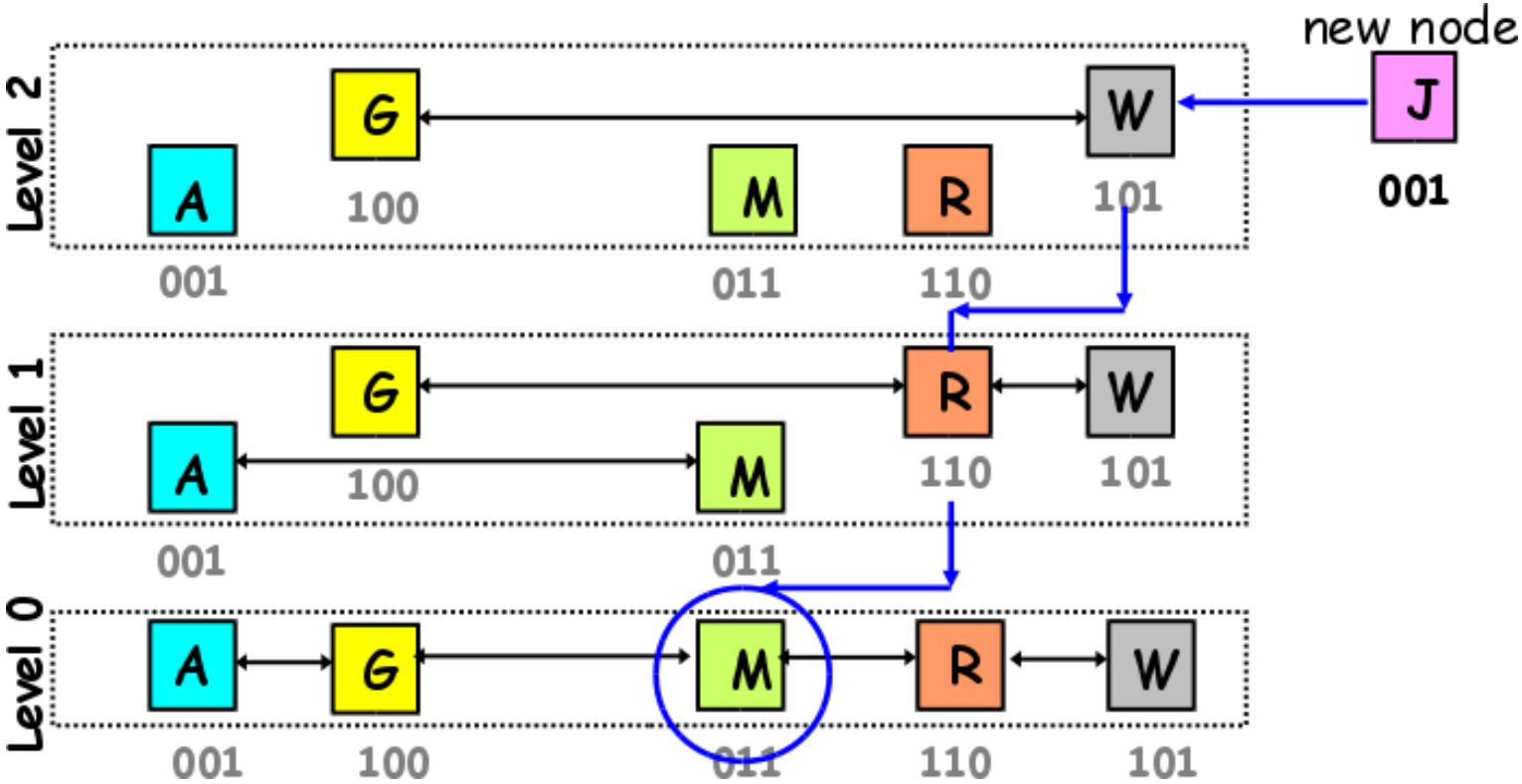


Algorithm 5 : *SkipGraph_Search* (Node n, Key k)

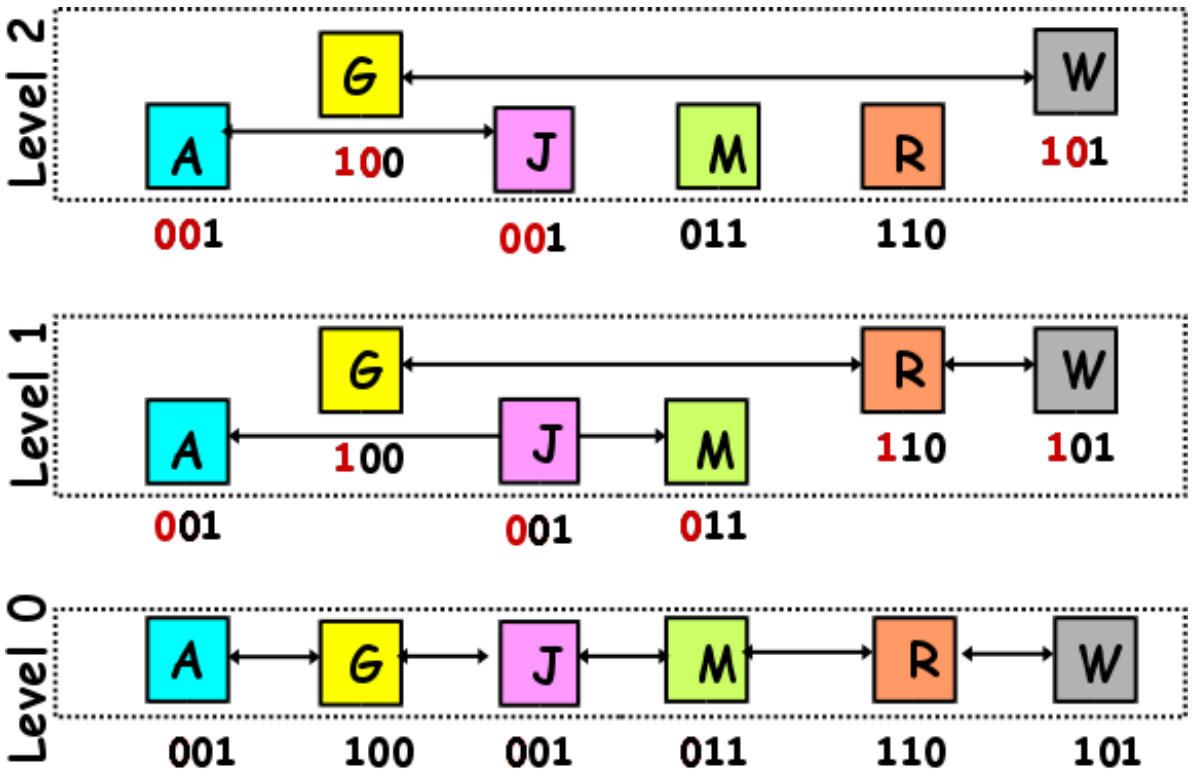
```
1: l = the highest level of n
2: while l ≥ 0 do
3:     find a neighbor node n' at level l that is closer to k
4:     if n' exists then
5:         SkipGraph_Search(n', k)
6:         return
7:     end if
8:     l = l − 1
9: end while
10: result = Local_Search(k)
11: return result to the query issuer node
```

Budování systému:

- při připojení nového uzlu s identifikátorem X:
  - na základě jeho vektoru příslušnosti $m(X)$ se uzel X připojuje do seznamů uzlů, jejichž vektor příslušnosti má shodný prefix s $m(X)$ (délka společného prefixu závislá na úrovni seznamu)
  - specifičtěji:
    - X se nejprve připojí do úrovně 0 na pozici, na kterou dle své klíčové položky (dat) náleží
    - pro každou úroveň $i \geq 1$ se X připojuje k nejbližšímu uzlu Y sdílejícímu s uzlem X shodný prefix o délce i



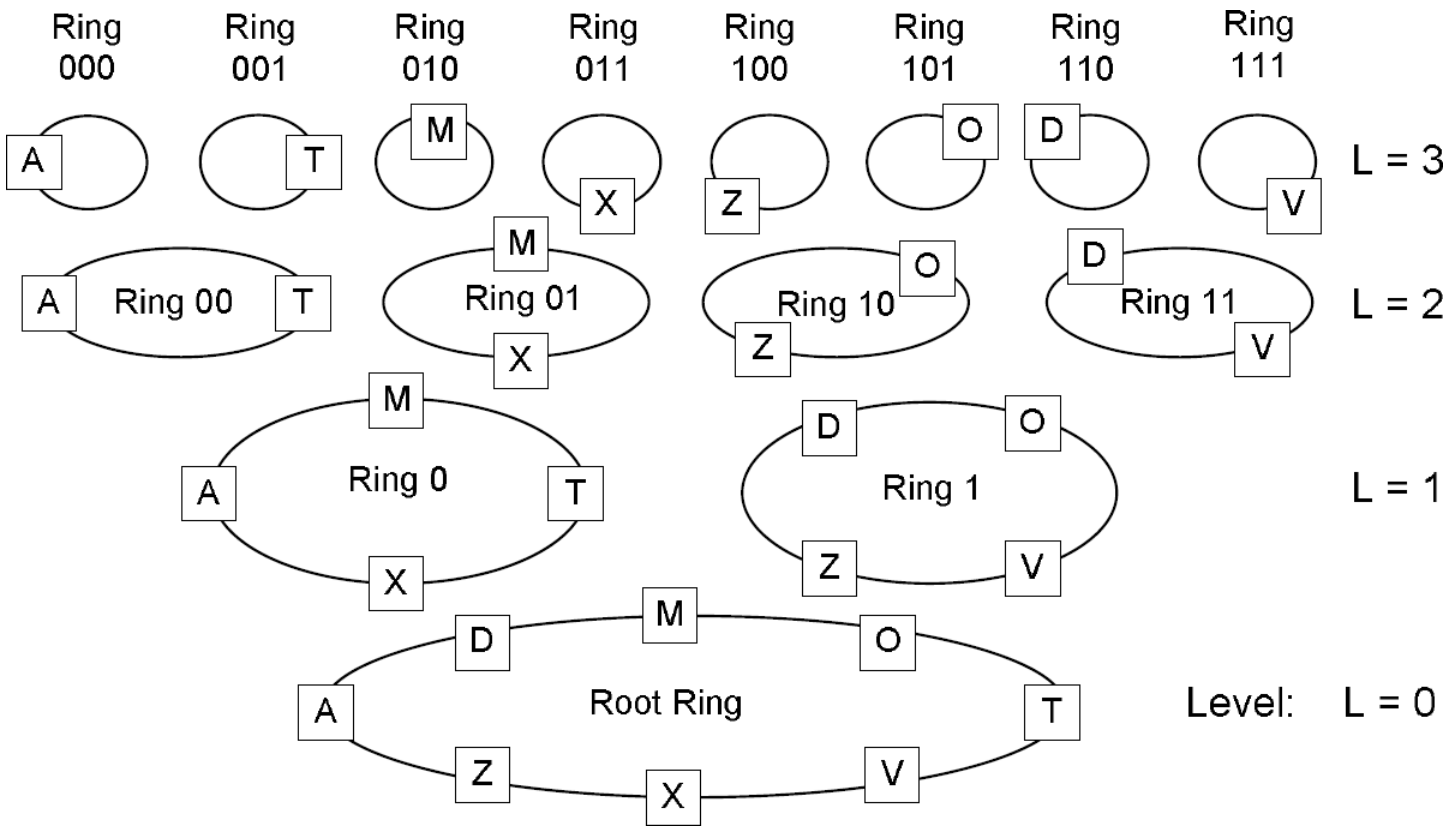
Obrázek: *Krok 1:* Začínaje v libovolném uzlu, najdi uzel s nejbližším **(datovým) klíčem** na úrovni 0.



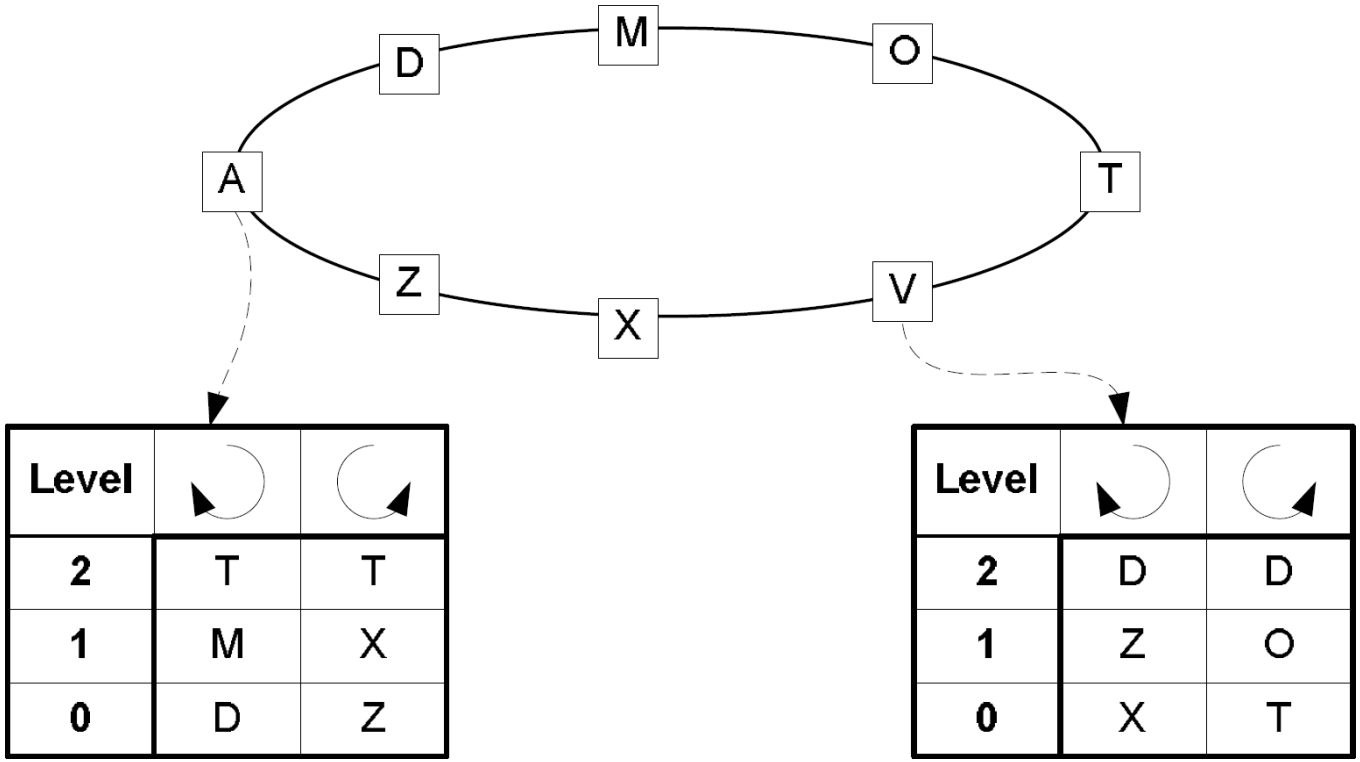
Obrázek: *Krok 2:* Na každé úrovni i se připoj do seznamu, s nímž sdílíš stejný prefix vektoru příslušnosti (o délce i).

idea:

- směrovací systém velmi podobný systému Skip Graph
- místo skip-listů jsou v rámci SkipNetu uzly organizovány do *kruhů*
  - organizovaných do úrovní/levelů (podobně jako u systému Skip Graph)
  - uzly jsou *na každé úrovni setřízeny podle datového klíče*
  - každý uzel má na každé ze svých úrovní ve směrovací tabulce uloženy dva ukazatele ke svým sousedům
    - ukazatele na úrovni h ukazují na uzly, které jsou zhruba 2^h uzlů nalevo a napravo od daného uzlu
    - všechny uzly jsou na nejnižší úrovni (úroveň 0) propojeny tzv. *kořenovým kruhem (root ring)*
- mechanismus vyhledávání a budování systému jsou podobné mechanismům představeným pro systém Skip Graph



Obrázek: Kompletní směrovací informace uzlu 8 v rámci SkipNet sítě (včetně identifikátorů jednotlivých kruhů).

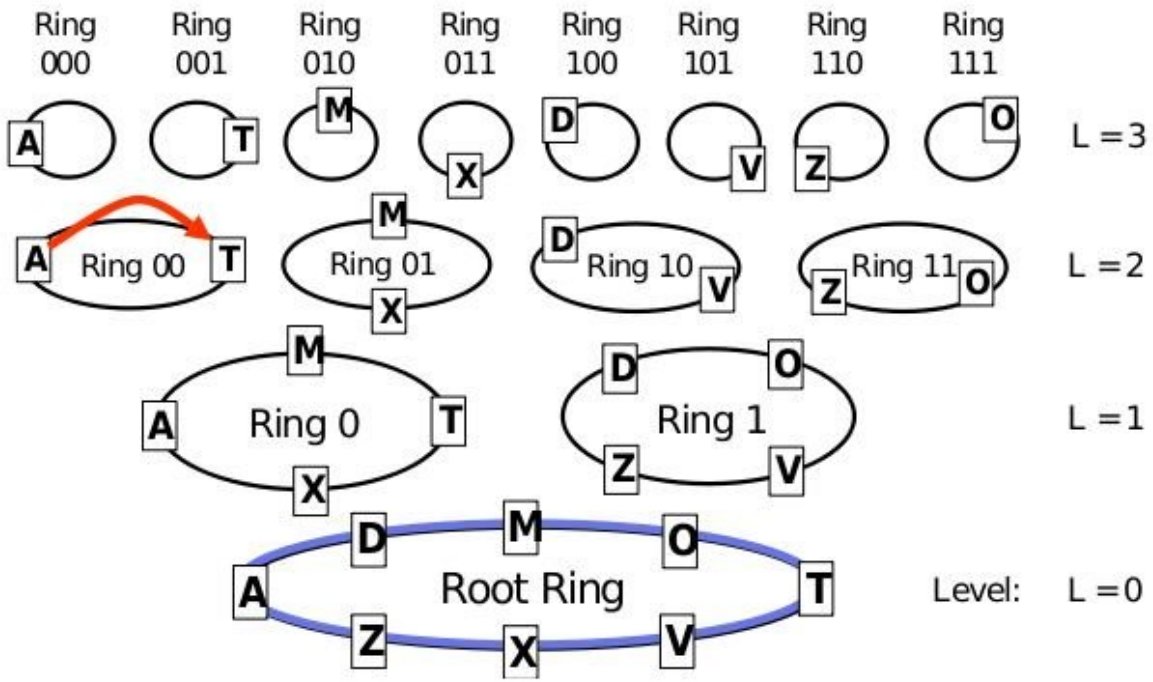


Level		
2	T	T
1	M	X
0	D	Z

Level		
2	D	D
1	Z	O
0	X	T

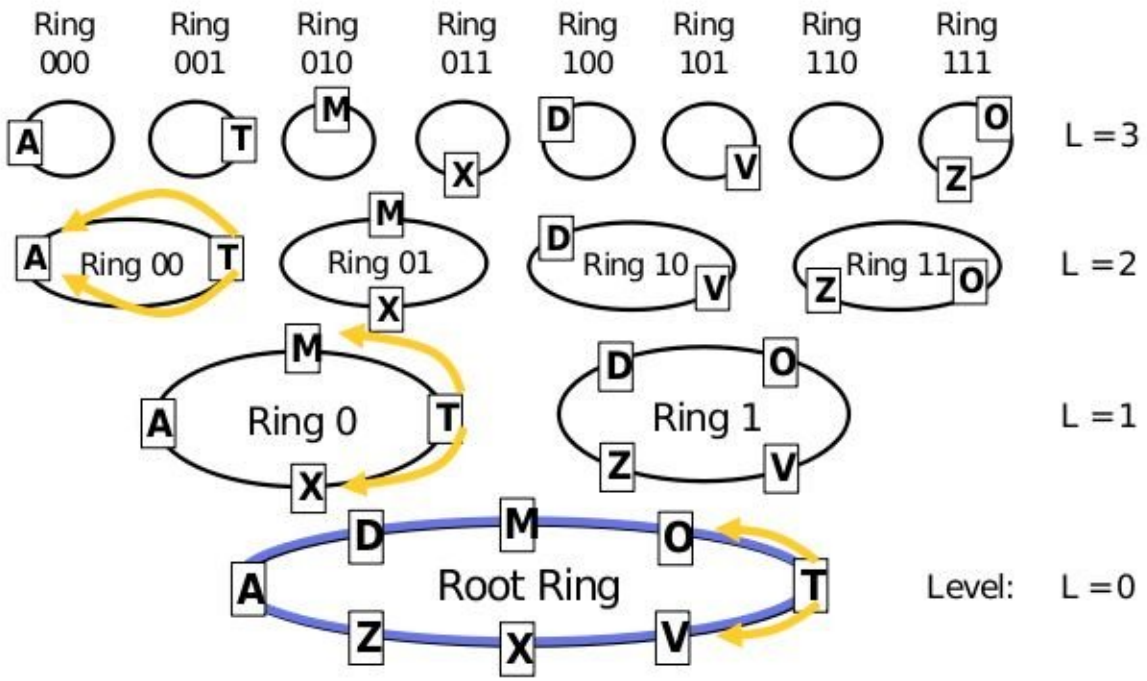
Obrázek: Směrovací tabulky pro uzly A a V.

Příklad vyhledávání: Nalezení uzlu V (počátek hledání v uzlu A)



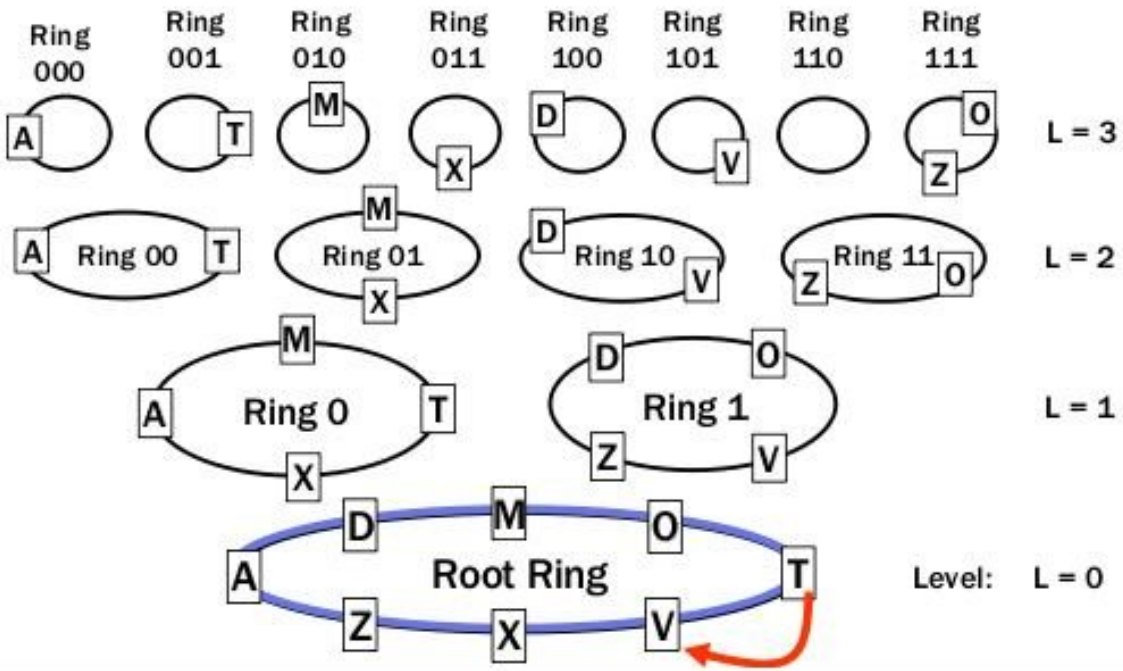
Obrázek: Zpráva je nejprve přeposlána uzlu T, neboť tento je blíže hledanému uzlu.

Příklad vyhledávání: Nalezení uzlu V (počátek hledání v uzlu A)



Obrázek: Směrovací tabulka uzlu T.

Příklad vyhledávání: Nalezení uzlu V (počátek hledání v uzlu A)

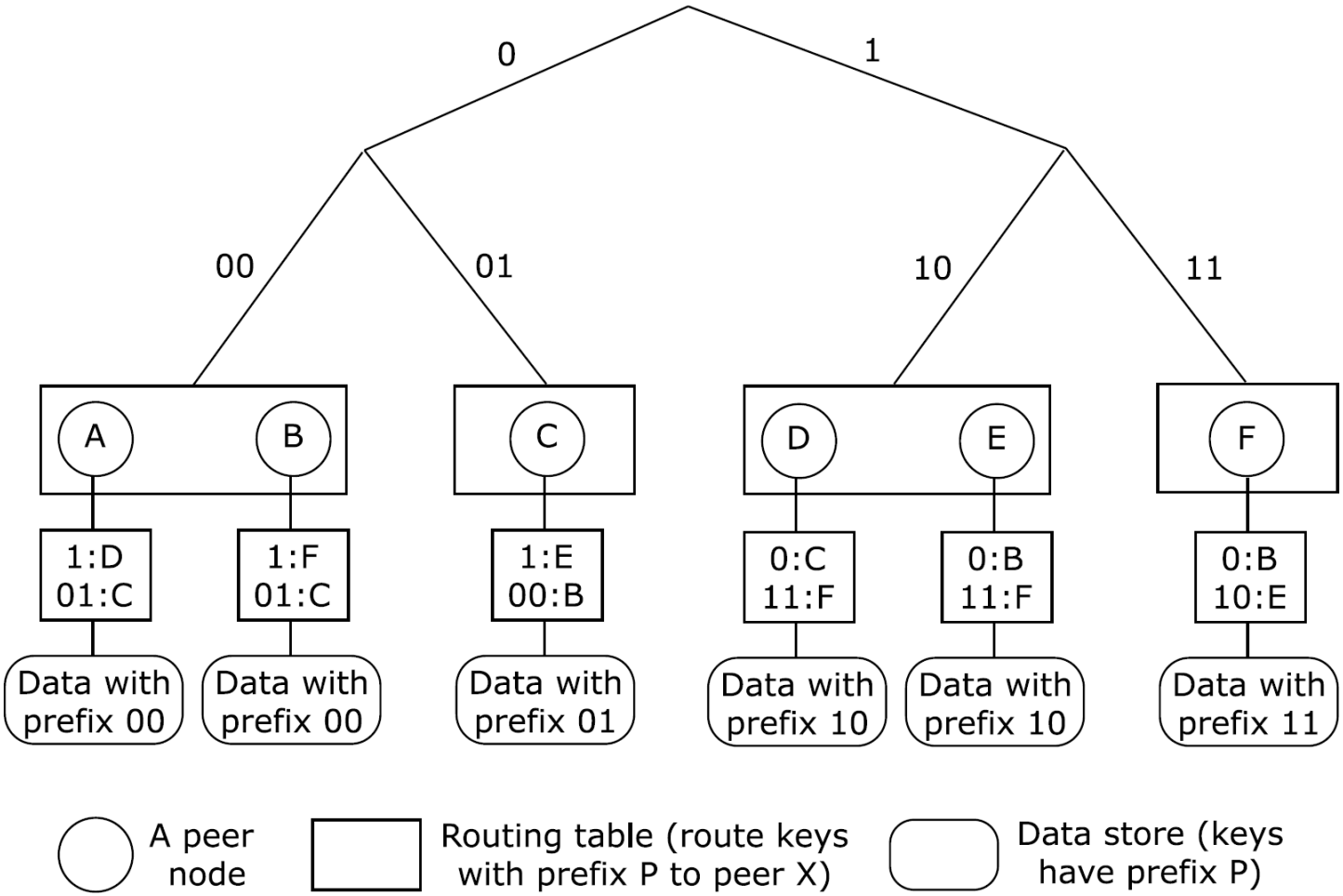


Obrázek: Jelikož má uzel T přímou informaci o hledaném uzlu V (na úrovni 0), vyhledávání končí (cíl je nalezen).

idea:

- systém P-Grid je založen na vybudovaném *virtuálním binárním stromu*, v rámci kterého každý z peerů spravuje jeden z listů daného stromu
- systém každému z peerů přiřazuje identifikátor, který je zřetězením binárních bitů reprezentujících cestu od kořene stromu k danému uzlu
- každý z peerů je pak zodpovědný za všechny datové položky, jejichž prefix identifikátoru klíče je shodný s identifikátorem daného uzlu
  - pro účely zvýšení odolnosti vůči výpadkům lze k jednomu identifikátoru přiřadit více peerů
- pro účely vyhledávání pak každý z peerů spravuje svou *směrovací tabulku*

Stromově-orientované směrovací strategie
P-Grid II.



mechanismus vyhledávání:

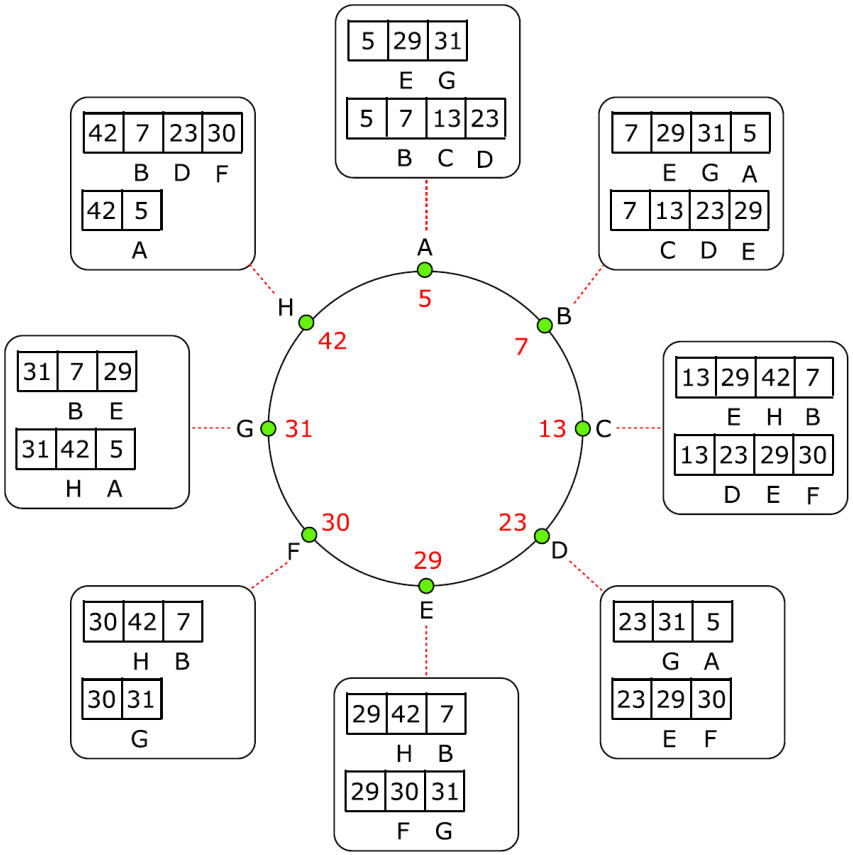
- jakmile uzel n přijme požadavek na vyhledání datové položky k, nejprve ověří, zda jeho identifikátor není prefixem k
  - pokud ano, vyhledává výsledek mezi jim spravovanými daty
  - pokud ne, využívá svou směrovací tabulku pro vyhledání peera, který je blíže vyhledávanému klíčí; tomu je pak požadavek přeposlán
- maximální počet vyhledávacích kroků je ohraničen hloubkou stromu
  - $\Rightarrow$ složitost vyhledávání je $O(log_2\, N)$

Algorithm 6 : *PGrid_Search* (Node n, Key k)

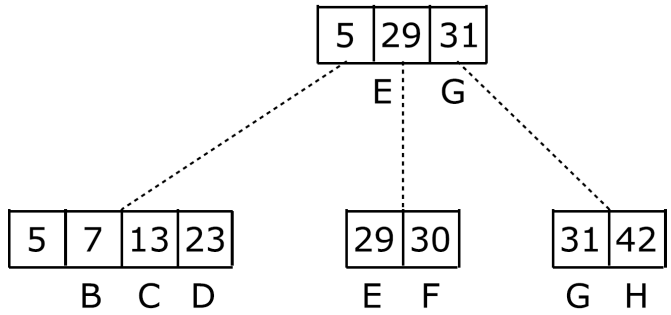
1: **if** n.id $\subseteq$ k **then**
2: result = Local_Search(k)
3: **return** result to the query issuer node
4: **else** {find a closer neighbor node to forward the query}
5: find l such that Prefix(k, l) = Invert(Prefix(n.id, l))
6: $n' = a$ randomly selected node from Routing(n) at level l
7: PGrid_Search(n', k)
8: **end if**

the idea:

- v rámci systému P-Grid nelze zaručit vyváženost stromové struktury
  - systém P-Tree je založen na *virtuálních vyvážených* B^+*-stromech* vybudovaných nad Chord strukturou
- každý z peerů spravuje:
  - uzel Chord sítě, který je listem dané stromové struktury, a
  - tzv. *semi-independent* B^+*-strom*, což je peerova představa o tzv. *fully independent* B^+*-stromu*
    - *fully independent* B^+*-strom* peera je B^+-strom, ve kterém hodnota klíče uloženého na daném uzlu je považována za nejmenší v dané Chord struktuře
    - *semi-independent* B^+*-strom* obsahuje pouze kořenové uzly a všechny uzly v nejlevější části daného fully-independent B^+-stromu
    - pro účely snazší správy se mohou rozsahy uzlů daných B^+-stromů překrývat (např. uzel C v následujícím obrázku)



(a) Semi-independent B^+-stromy spravované P-Tree uzly.



(b) Fully-independent B^+-strom uzlu A.

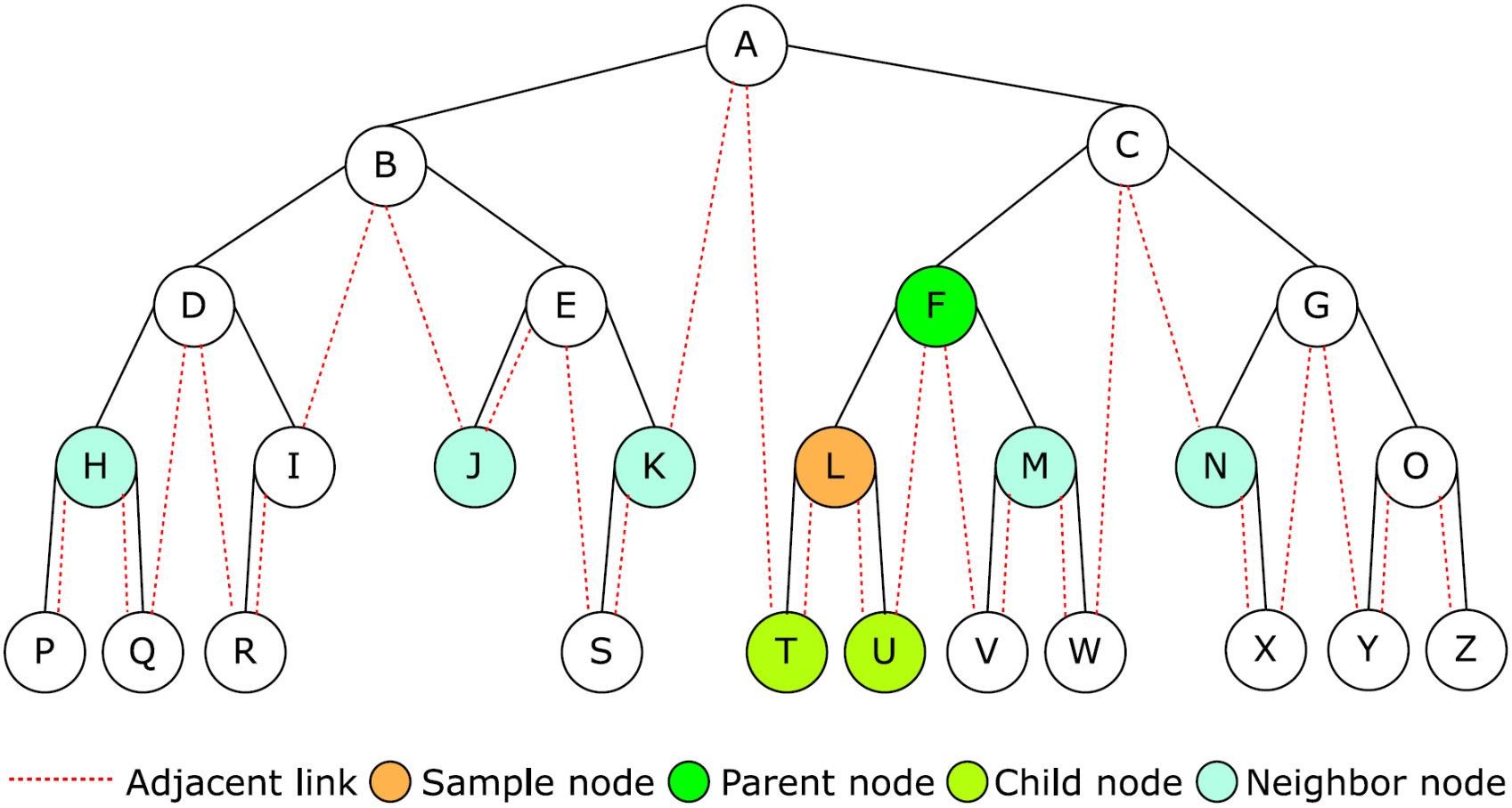
Algorithm 7 : PTree_Search(Node n, Range_Query q)

1: Find a neighbor node n' at the lowest level l and the maximum number j such that n'.value $\in$ (n.value, q.lowerBound)
2: **if** n' exists **then**
3: PTree_Search(n', q)
4: **else**
5: **if** n covers the search key **then**
6: result = Local_Search(q)
7: **return** result to the query issuer node
8: **end if**
9: n' = successor of n
10: **if** n'.value $\in$ (n.value, q.upperBound) **then**
11: PTree_Search(n', q)
12: **end if**
13: **end if**

idea:

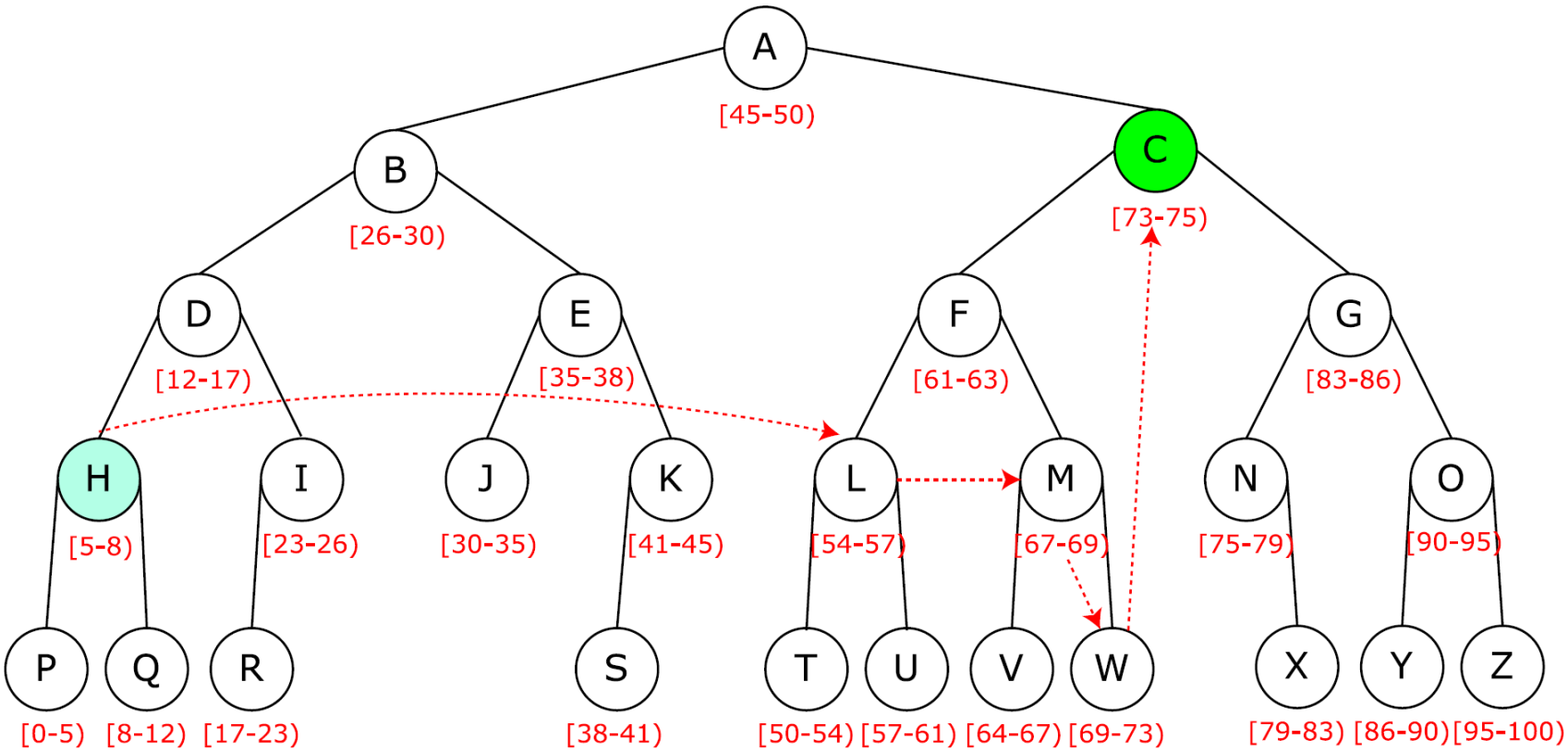
- oproti standardním stromovým strukturám se systém BATON odlišuje zejména dvěma vlastnostmi:
  - data jsou uložena jak na listech stromu, tak na vnitřních uzlech
  - kromě linek k rodičům a potomkům jsou na každém z uzlů udržovány ještě tzv. *linky k přilehlým uzlům (adjacent links)* a *linky k sousedům (neighbor links)*
    - *adjacent links* jsou využity pro propojení daného uzlu s uzlem/uzly, které spravují jemu přilehlé rozsahy klíčů
    - *neighbor links* jsou využity pro propojení daného uzlu se sousedy (na stejné úrovni stromu), kteří se od něj nalézají ve vzdálenosti 2^i, $i \geq 0$
    - účelem těchto linek je zabránění zahlcení kořene (resp. kořenového uzlu) vyhledávacími dotazy

Stromově-orientované směrovací strategie
BATON II.



detaily vyhledávání:

- jakmile uzel x přijme požadavek na vyhledání:
  1. jestliže hledaný klíč spadá do jim spravovaného rozsahu, odpovídá na dotaz on sám
  2. jinak požadavek přeposílá svému nejvzdálenějšímu sousedovi (na stejné úrovni), který spravuje k vyhledávanému klíči bližší (avšak nepřevyšující) záznamy
  3. pokud takovýto uzel neexistuje, přeposílá uzel x požadavek na vyhledání buď svému potomkovi (pokud existuje) nebo přilehlému uzlu ve směru vyhledávání (tj. s využitím tzv. *adjacent linky*)



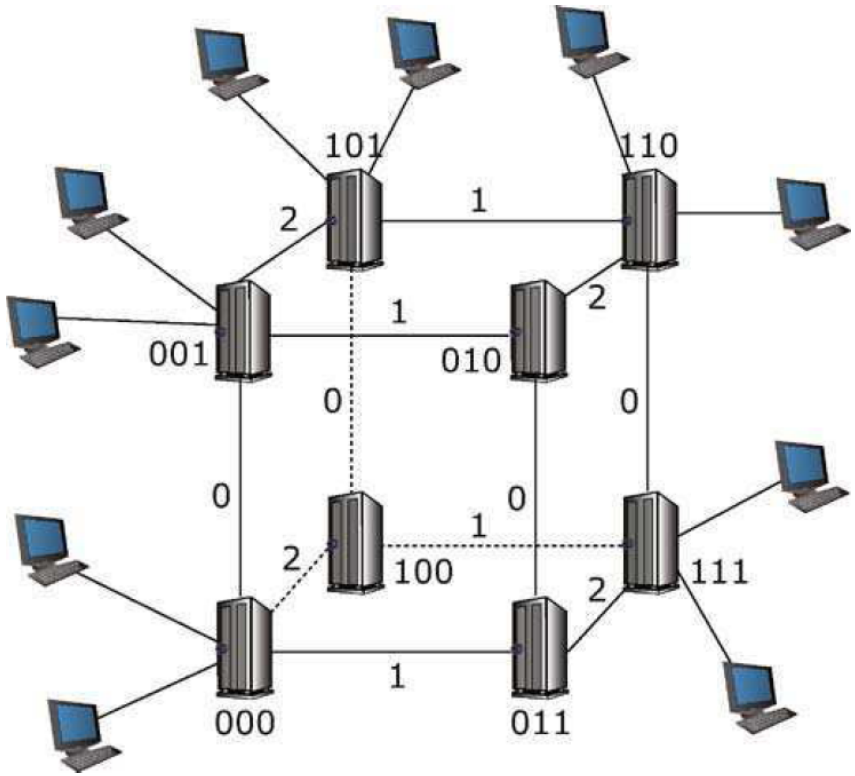
Obrázek: Příklad vyhledávání v systému BATON: uzel *H* vyhledává datovou položku (s klíčem 74) uloženou na uzlu *C*.

Algorithm 8 : BATON_Search(Node n, Key k)

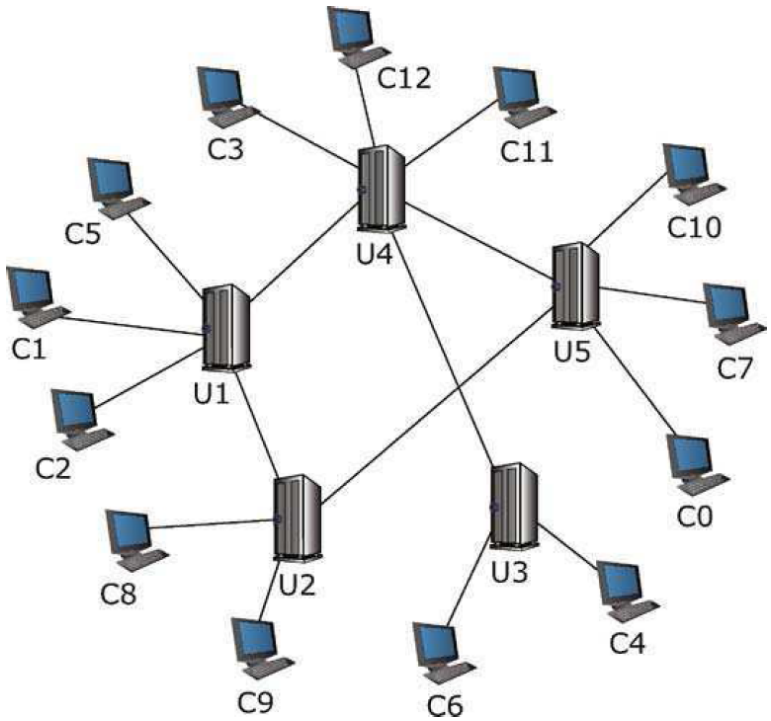
```
1:  if n covers the search key then
2:      result = Local_Search(q)
3:      return result to the query issuer node
4:  else
5:      if there exists a neighbor n' of n that is closer to k then
6:          BATON_Search(n', k)
7:      else
8:          if there exists a child n' of n that is closer to k then
9:              BATON_Search(n', k)
10:         else
11:             n' is an adjacent node of n that is closer to k
12:             BATON_Search(n', k)
13:         end if
14:     end if
15: end if
```

Směrování v hybridních P2P sítích

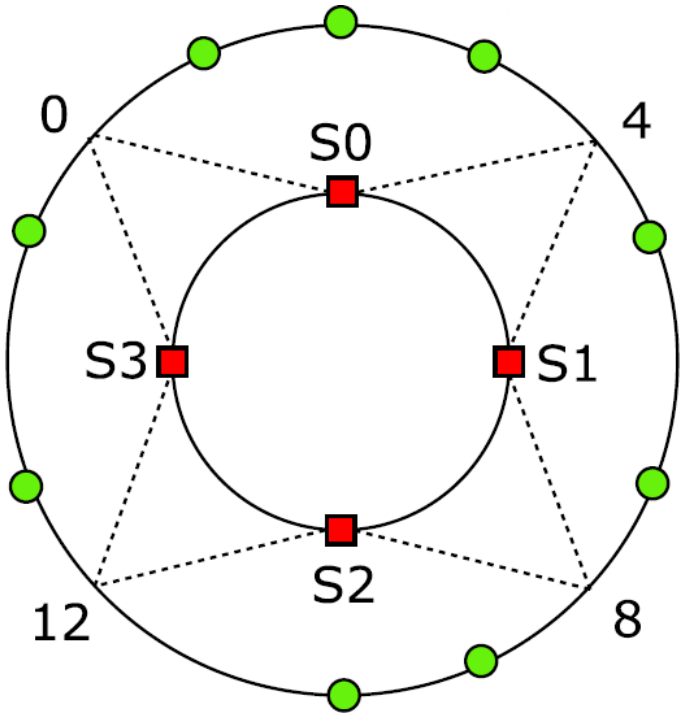
- hybridní P2P systémy organizují peery do hierarchických struktur
  - vybrané (mocné, *powerful*) uzly (*superpeers*, *supernodes*) jsou umístěny do vyšší úrovně, a
  - běžné peer uzly (často nazývané taky *klientské uzly*) jsou umístěny do nižších úrovní
  - každý klientský uzel náleží k jednomu superpeer uzlu a nekomunikuje s ostatními uzly jinak než přes svého superpeera
- obecné směrovací schéma hybridních P2P sítí:
  1. klientský uzel zasílá požadavek na vyhledání svému superpeer uzlu
  2. superpeer prohledává svůj adresář za účelem zjištění, který jemu náležející klientský uzel či který jiný superpeer (do jehož oblasti spadá hledaný uzel) obsahuje odpověď na hledaný dotaz
  3. požadavek je přeposlán superpeerovi, který by měl mít požadovanou odpověď
     - tento superpeer pak s využitím svého adresáře informací o jim spravovaných klientech vyhledává peera, který obsahuje hledaná data
  4. IP adresa vyhledaného peera je pak zaslána zdrojovému uzlu
     - následná komunikace (získání dat) pak probíhá napřímo mezi zdrojovým a vyhledaným uzlem
- příklady hybridních sítí:
  - KaZaA, BestPeer, Edutella, atd.



Obrázek: **Síť Edutella.** Požadavek na vyhledání je nejprve směrován na superpeery, kteří jsou organizováni do tzv. in *HyperCuP sítě* (= HyperCube P2P), v rámci které může být využito směrovací schéma na bázi nejdelšího společného suffixu.



Obrázek: **Modifikovaná síť Gnutella s ultrapeery.** Předpokládejme, že data požadované peerem $C12$ jsou umístěny na peerovi $C9$: peer $C12$ nejprve zasílá svůj požadavek na vyhledání svému ultrapeerovi $U4$, který s využitím mechanismu záplavy rozesílá požadavek do sítě, který se tak skrze uzel $U1$ dostane na uzel $U2$; uzel $U2$ prohledává svůj index jim spravovaných uzlů a zjišťuje, že požadovaná data jsou na uzlu $C9$ $\rightarrow$ zasílá IP adresu uzlu $C9$ dotazujícímu se uzlu $C12$.



Obrázek: **Structured superpeers:** superpeer uzly $S0$, $S1$, $S2$, a $S3$ spravují (po řadě) rozsahy klíčů $(0, 4]$, $(4, 8]$, $(8, 12]$ a $(12, 0]$. Pokud např. peer $P1$ vyhledává data s klíčem $key = 10$, nejprve zasílá požadavek na vyhledání svému superpeer uzlu $S0$; $S0$ předává dotaz uzlu $S2$ (jelikož $S2$ spravuje rozsah, kam vyhledávaná položka náleží), který pak uzlu $P1$ odpovídá s informací o IP adrese jim nalezeného uzlu, který vyhledávaná data obsahuje.

	structured P2P	unstructured P2P
routing	based on a routing table	flooding, random walk, ...
lookup possibilities	based on keys only	possibility to ask more compex queries
existing item is always found	yes	cannot be guaranteed
critical part	node join/disconnect	lookup/routing

Celkové shrnutí přednášky

- Grafy velmi důležitou diskrétní konstrukcí matematiky i informatiky
- Řada reálných problémů převeditelná na grafové problémy
  - Dobrá znalost alespoň základů teorie grafů pomůže
  - Znalost grafových algoritmů resp. problémů může pomoci při odhadu náročnosti řešení konkrétního praktického problému
  - Mapování na grafový problém může najít optimální řešení
- Plánování jako grafový problém
- Velmi významné optimalizace