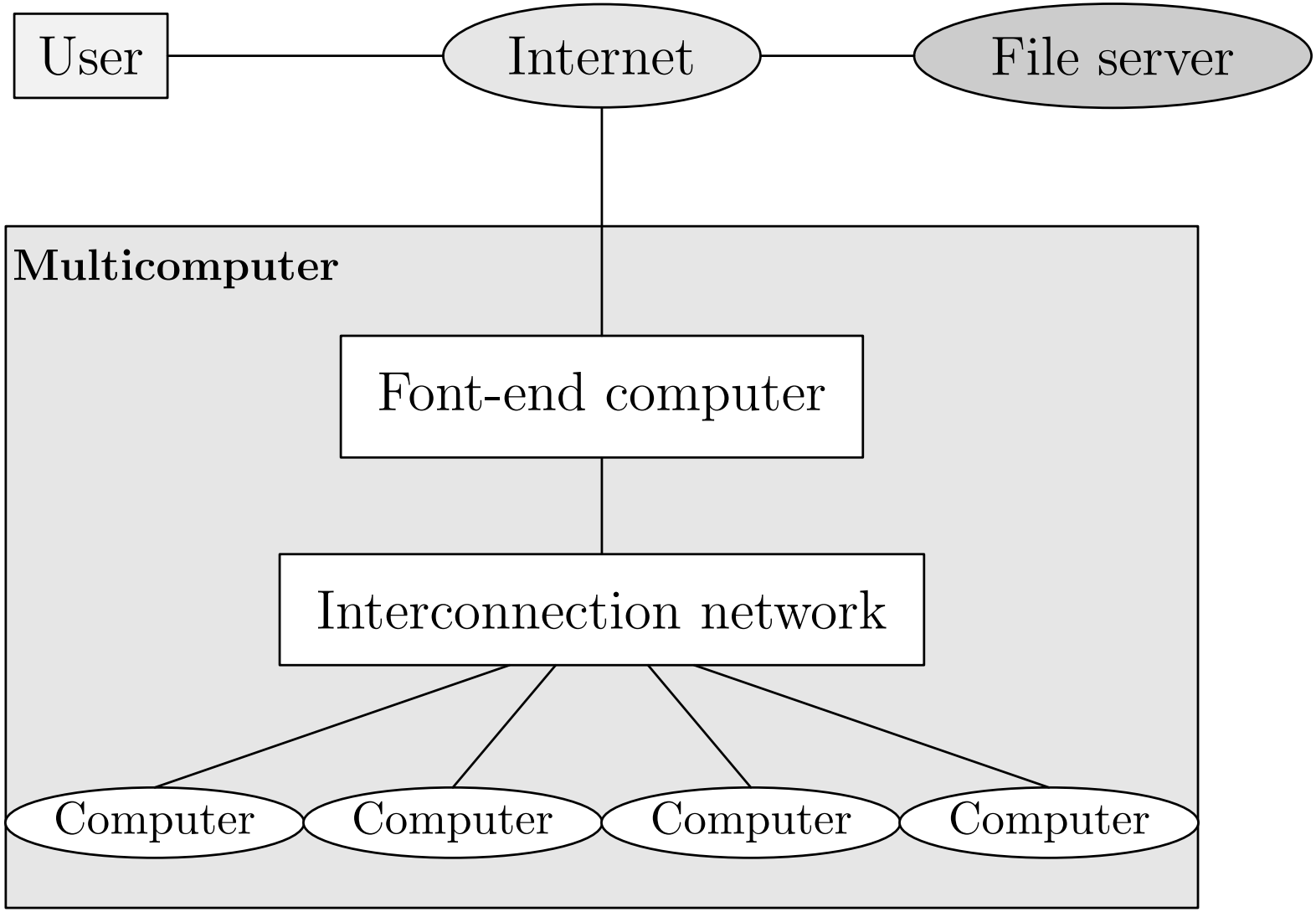
Architektury s distribuovanou pamětí

Multipočítače

- *multicomputer* je podobný multiprocesoru se sdílenou pamětí, ale nemá společný adresový prostor
- každý procesor má přístup je do své vlastní paměti
- stejná adresa u dvou různých procesorů odpovídám různým paměťovým buňkám
- procesory spolu komunikují pouze posíláním zpráv přes komunikační síť, která je propojuje
- dělí se na
  - **nesymetrické**
  - **symetrické**

Nesymetrické multipočítače I.



Nesymetrické multipočítače II.

- první multipočítače byly asymetrické
- měly jeden počítač (*front-end computer*) pro komunikaci s uživateli a s I/O zařízeními
- dále obsahovaly řadu počítačů určených pouze pro výpočty - *back-end*
- tyto výpočetní počítače jsou vybaveny velmi malým a jednoduchým operačním systémem
  - jsou bez multitaskingu, virtuální paměti, I/O rozhraní
- příklady
  - Intel iPSC - 128 uzlů s operačním systémem NX
  - nCUBE/ten - Intel 80286 (font), 1024 uzlů s operačním systémem VERTEX

Nesymetrické multipočítače III.

Výhody nesymetrických multipočítačů:

- jednoduché výpočetní uzly jsou levnější, než plnohodnotné počítače
- jednoduchý operační systém výpočetních uzlů umožňuje optimálnější využití hardwaru
  - multitasking může výrazně zpomalovat

Nesymetrické multipočítače III.

Nevýhody nesymetrických multipočítačů:

- pokud selže přístupový počítač (*front-end*), je celý multipočítač nepřístupný
- škálovatelnost je omezena výkonem komunikační sítě a přístupového počítače
  - multipočítače jsou víceuživatelské
  - *front-end* je používán k ladění, kompilování, I/O operacím
  - při větším počtu uživatelů to může vést k přetížení
  - lze to částečně řešit přidáním dalšího *front-endu* - to ale není příliš elegantní
- primitivní operační systémy na výpočetních uzlech znesnadňují ladění kódu
  - *back-end* nedokáže jednoduše vypisovat ladicí zprávy
  - je nutné poslat po síti zprávu *front-endu*, který jí pak vypíše
- je nutné psát zvláštní kód pro *front-end* a *back-end*

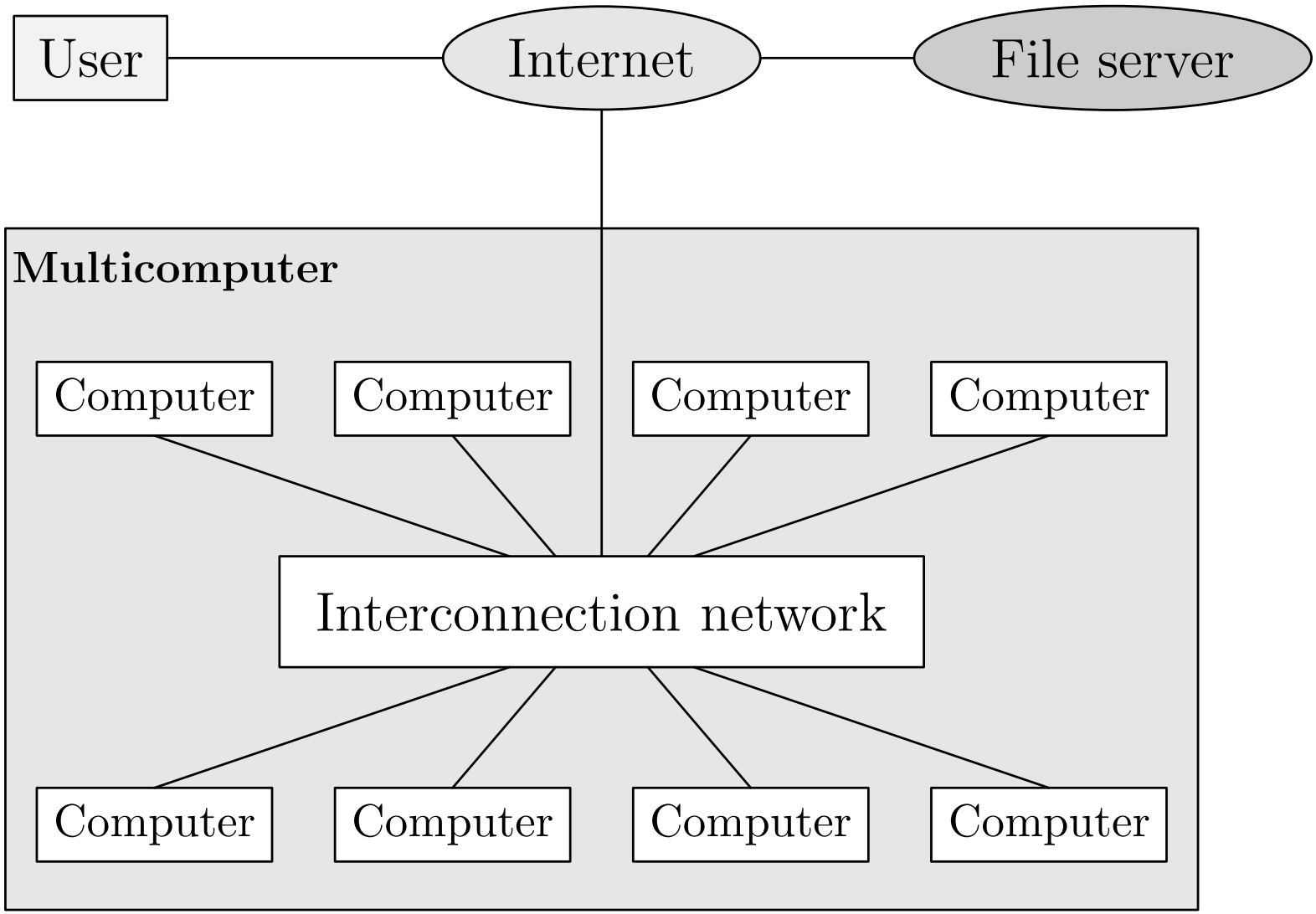
Nesymetrické multipočítače III.

Nevýhody nesymetrických multipočítačů:

- pokud selže přístupový počítač (*front-end*), je celý multipočítač nepřístupný
- škálovatelnost je omezena výkonem komunikační sítě a přístupového počítače
  - multipočítače jsou víceuživatelské
  - *front-end* je používán k ladění, kompilování, I/O operacím
  - při větším počtu uživatelů to může vést k přetížení
  - lze to částečně řešit přidáním dalšího *front-endu* - to ale není příliš elegantní
- primitivní operační systémy na výpočetních uzlech znesnadňují ladění kódu
  - *back-end* nedokáže jednoduše vypisovat ladicí zprávy
  - je nutné poslat po síti zprávu *front-endu*, který jí pak vypíše
- je nutné psát zvláštní kód pro *front-end* a *back-end*

Zejména poslední dva body vedly k přechodu k symetrickým multipočítačům.

Symetrické multipočítače I.



Symetrické multipočítače II.

- u symetrického multipočítače jsou všechny počítače rovnocenné
- na každém počítače běží stejný, plně funkční operační systém
- uživatel se může přihlásit na libovolný uzel
- programátor nemusí rozlišovat mezi kódem pro přístupový a výpočetní uzel
  - má-li některý počítač provést odlišný kód, řeší se to `if-then-else` konstrukcí

Symetrické multipočítače III.

Nevýhody symetrických multipočítačů

- nevytváří dojem jednotného paralelního počítače, ale spíše shluku propojených počítačů
  - lze k nim přistupovat přes více IP adres
- pokud každý počítač slouží jako *front-end* tj. k ladění a kompilování, je těžké vybalancovat zatížení jednotlivých uzlů
- uzly, které by měly sloužit jen k výpočtům, jsou zatěžovány obsluhou uživatelů

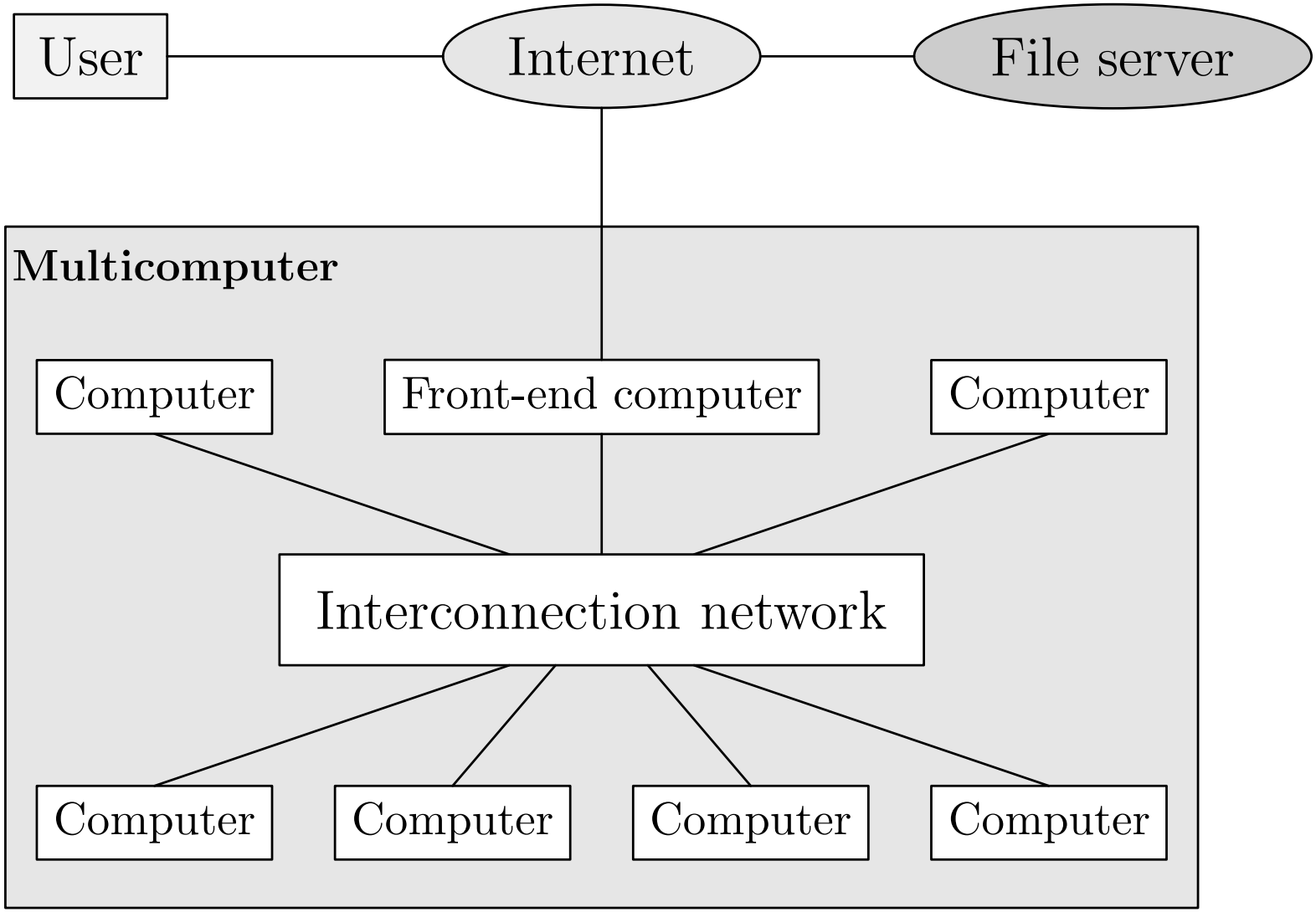
Symetrické multipočítače III.

Nevýhody symetrických multipočítačů

- nevytváří dojem jednotného paralelního počítače, ale spíše shluku propojených počítačů
  - lze k nim přistupovat přes více IP adres
- pokud každý počítač slouží jako *front-end* tj. k ladění a kompilování, je těžké vybalancovat zatížení jednotlivých uzlů
- uzly, které by měly sloužit jen k výpočtům, jsou zatěžovány obsluhou uživatelů

Za účelem odstranění těchto problémů se symetrický multipočítač doplní přístupovým uzlem - *front-endem*.

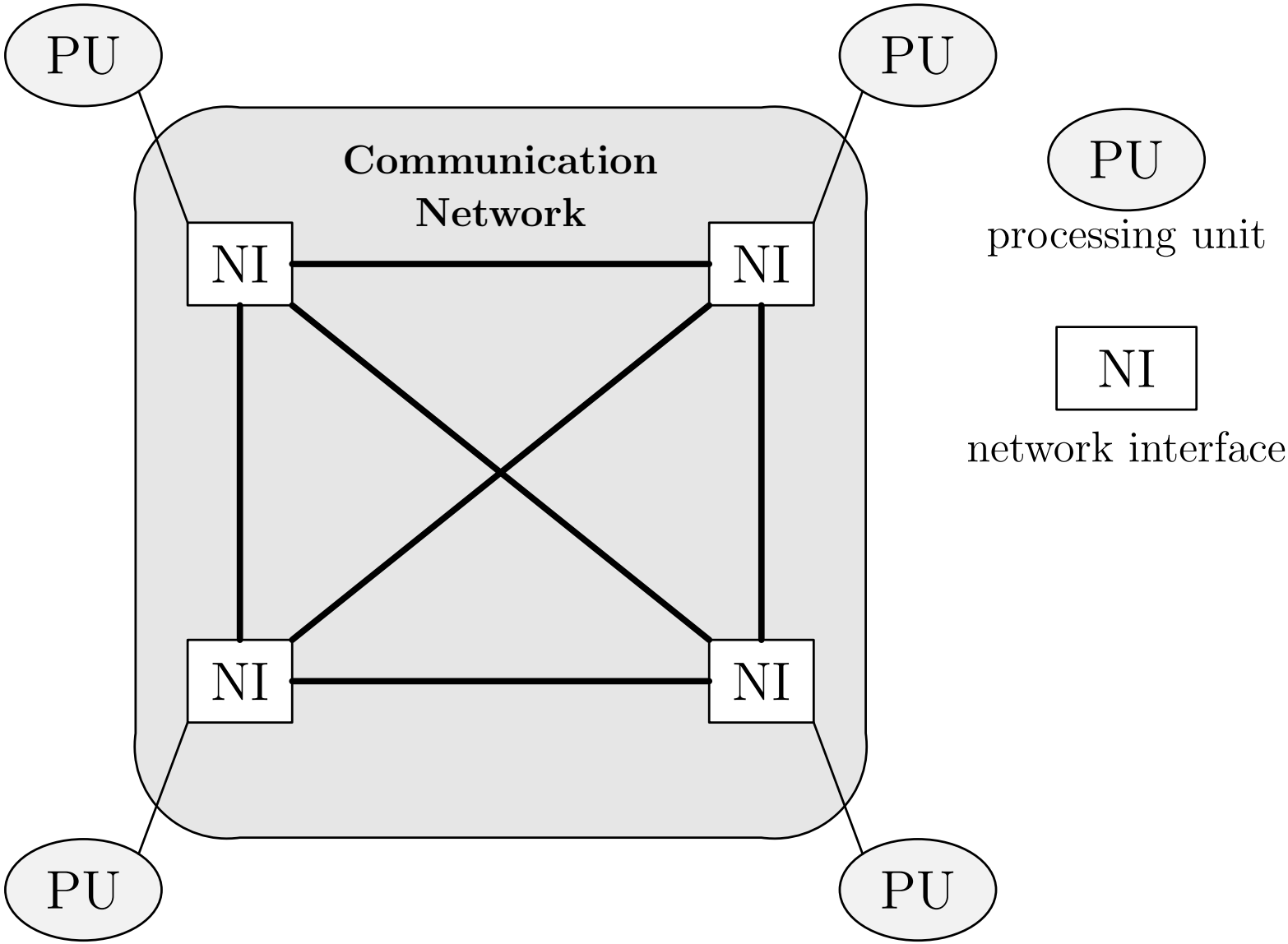
Symetrické multipočítače IV.



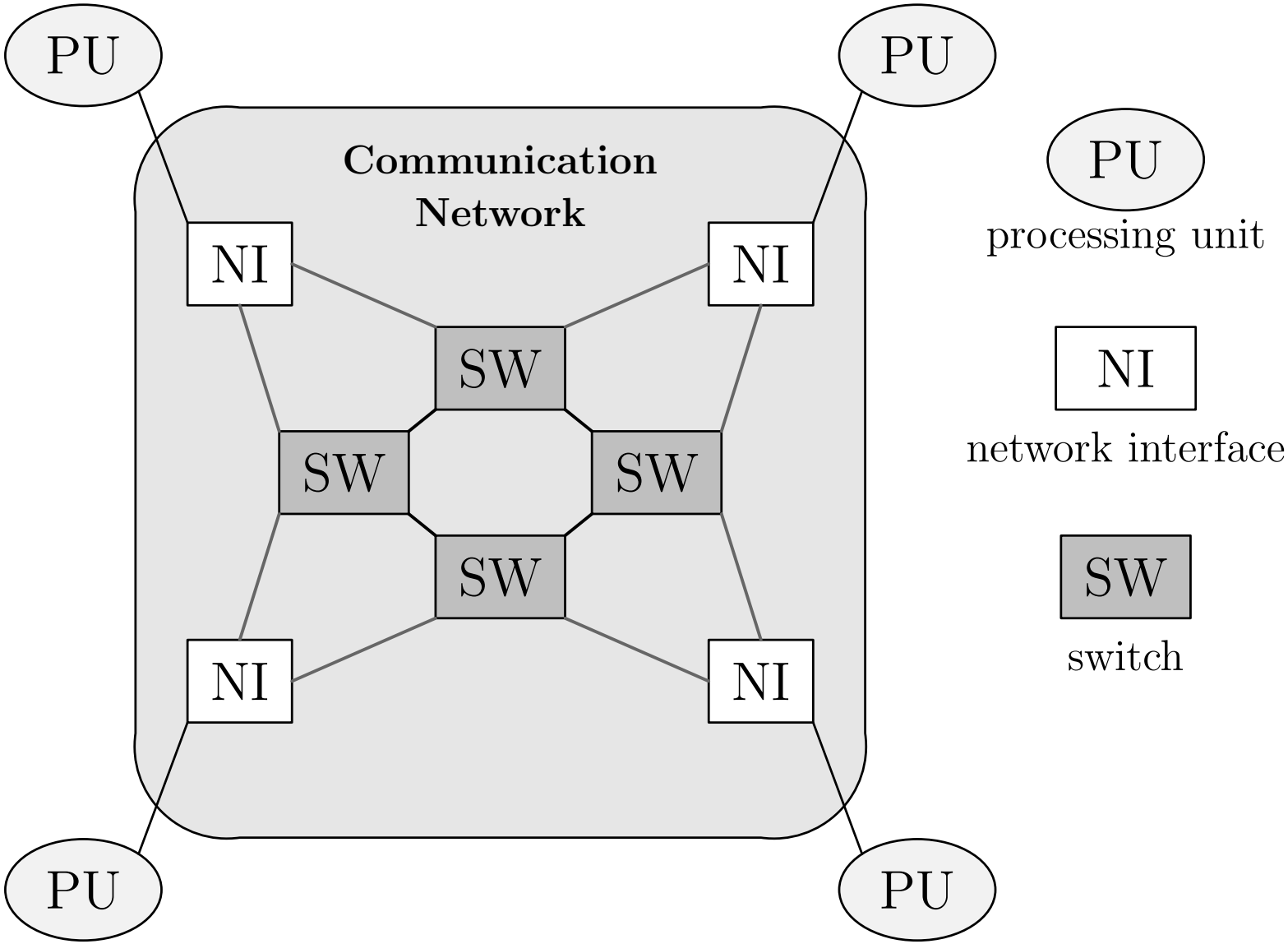
Symetrické multipočítače IV.

- protože jednotlivé uzly multipočítače spolu nesdílí paměť, výpočty na nich se dělí na procesy (ne vlákna)
- komunikace mezi procesy probíhá pomocí komunikační sítě
- následně si probereme, jaké topologie sítí se používají a jak probíhá síťová komunikace

Přímá komunikační síť



Nepřímá komunikační síť



Síťový přepínač - *switch*

- provádí mapování (přepínání) mezi vstupy a výstupy = **porty**
- počet všech portů switche se nazývá **stupeň** - *degree*
- switch může mít
  - vnitřní vyrovnávací paměť - *buffer*
  - hardware pro směrování dat v síti - *router*
  - podporu přesměrování jednoho vstupu na více výstupů - *multicasting*

Síťové rozhraní - *network interface*

- vytváří datové pakety
  - rozdělí data na menší celky
  - přidá hlavičku s doprovodnými informacemi
  - provádí opravu chyb

Škálovatelnost I.

- **škálovatelnost** (*scalability*) systému je vlastnost popisující, jak snadno lze měnit velikost systému
- nejčastěji nás zajímá, jak složité/nákladné je systém zvětšit v případě, že současný stav nevyhovuje našim požadavkům
- je-li systém dobře škálovatelný, lze ho snadno přizpůsobit našim potřebám
- v případě paralelních architektur je škálovatelnost dána typem komunikační sítě
  - cena za nový uzel je vždy stejná
  - s rostoucím počtem uzlů roste i počet nutných spojů a případně i přepínačů
- zajímá nás vztah mezi počtem výpočetních jednotek p a náklady na jejich propojení
- ten vyjadřujeme asymptoticky pomocí tzv. **Bachmannova-Landauova značení**

Škálovatelnost II.

Bachmannovo-Landauovo značení

Pro funkce $f, g : \mathbb{R}^+ \to \mathbb{R}^+$ definujeme

- $f(n) \in O(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq C$
  - f je nejvýše řádu g
- $f(n) \in \Omega(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} \geq C$
  - f je alespoň řádu g
- $f(n) \in \Theta(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = C$
  - f je stejného řádu jako g
- $f(n) \in o(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
  - f je řádu ostře menšího než g
- $f(n) \in \omega(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$
  - f je řádu ostře většího než g

Přímé komunikační sítě

- komunikační sítě multipočítačů odpovídají nejčastěji přímým sítím
- síťová spojení přímých sítí jsou dána pevně a nemění se
- přímé sítě se popisují pomocí grafů

Definition

Neorientovaný graf je dvojice $G(V, E)$ tvořená neprázdnou množinou **vrcholů** V a konečnou množinou **hran** E.

Definition

Jsou-li $v_1, v_2 \in V$ dva vrcholy a $e \in E$ taková, že $e = (v_1, v_2)$, říkáme, že hrana e spojuje vrcholy v_1 a v_2.

- vrcholy představují výpočetní jednotku (uzel) - *computing node*
- hrany představují síťová spojení - *link*

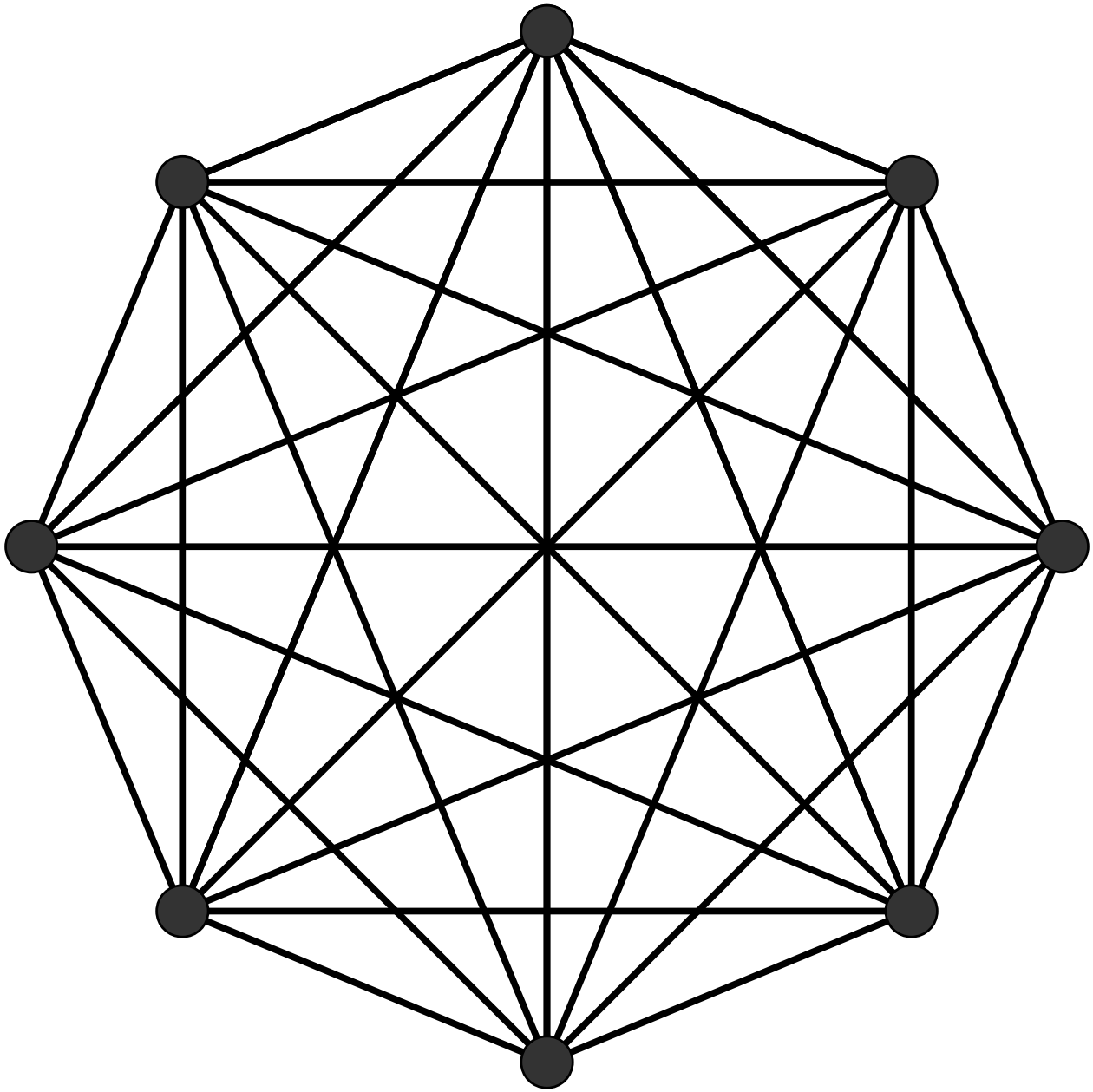
Úplně propojená síť - *completely connected network*

Tento typ sítě odpovídá úplnému grafu.

Definition

Úplný graf je takový graf, jehož libovolné dva uzly jsou spojeny hranou.

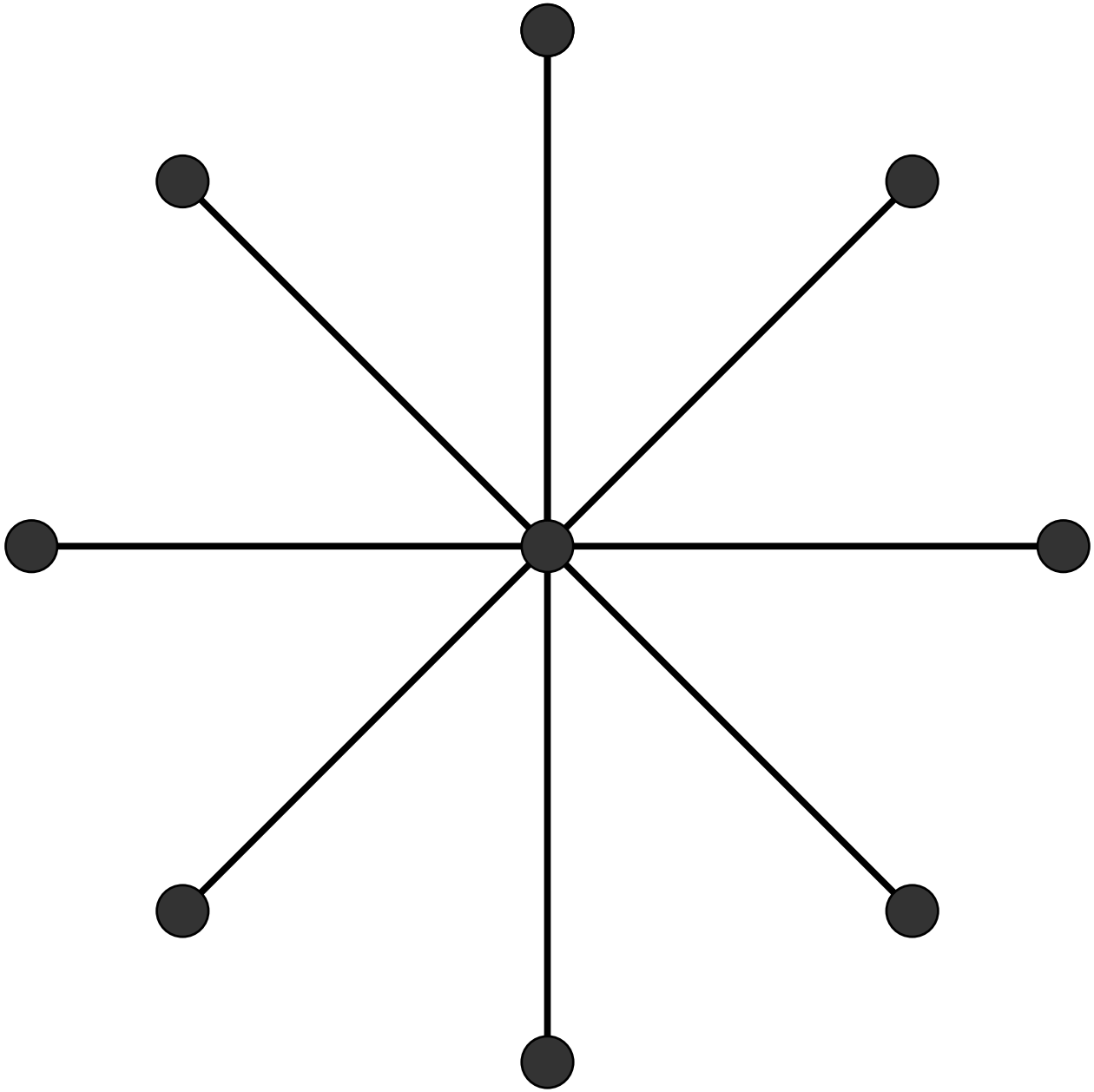
Úplně propojená síť - *completely connected network*



Úplně propojená síť - *completely connected network*

- v libovolný okamžik může komunikovat libovolný uzel s libovolným jiným
- síť má $p(p-1)/2$ spojů $\Rightarrow$ náklady jsou řádu $\theta(p^2)$
- úplně propojená síť je tudíž špatně škálovatelná

Hvězdicovitá síť - *star connected network*



Hvězdicovitá síť - *star connected network*

- má jeden centrální uzel, který zajišťuje komunikace mezi ostatními
- centrální počítač je ale (stejně jako komunikační médium u sběrnice) úzkým hrdlem této sítě
- síť obsahuje p komunikačních spojů
- náklady jsou tedy řádu $\theta(p)$

Ortogonální sítě - *orthogonal networks*

- jde o sítě, jejichž topologie je podobná topologii euklidovských prostorů $\mathbb{R}^n$
- tyto sítě se nejlépe popisují rekurzivně pomocí součinu grafů

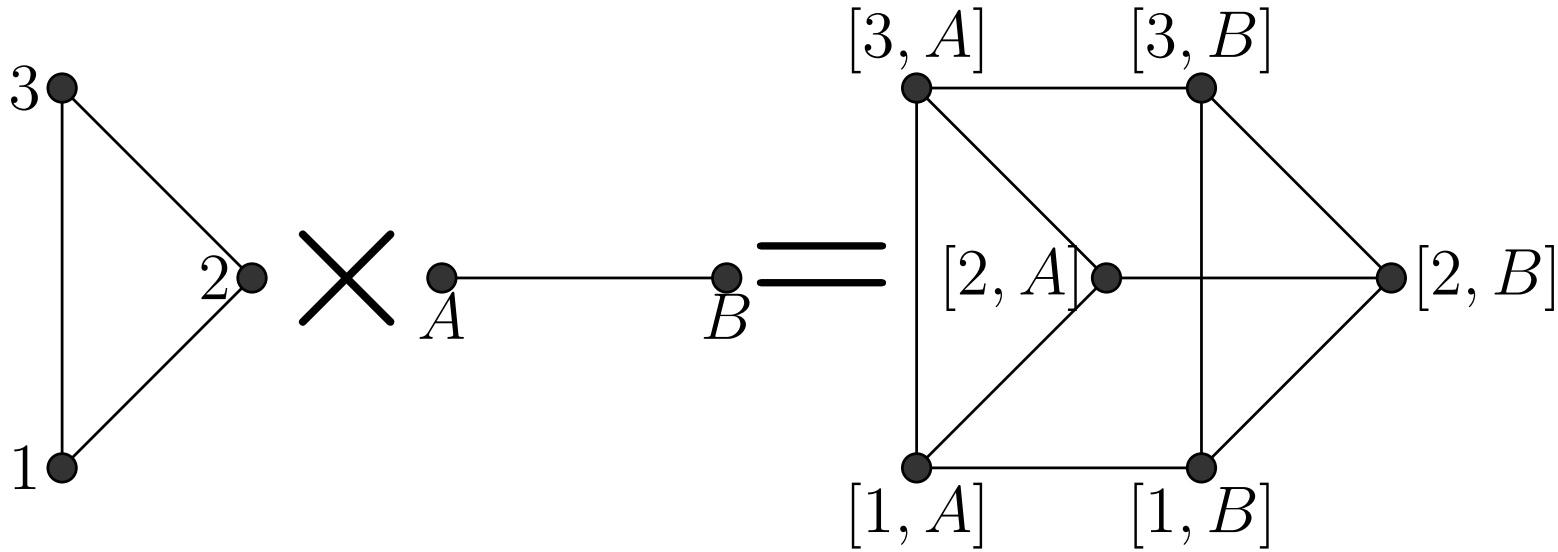
Definition

Kartézský součin dvou různých grafů $G_1 = G_1(V_1, E_1), G_2 = G_2(V_2, E_2)$ je graf $G = G_1 \times G_2$, $G = G(V, E)$, pro který platí

$$\begin{aligned} V &= \{[x, y] \mid x \in V_1; y \in V_2\}, \\ E &= \{([x_1, y], [x_2, y] \mid (x_1, x_2) \in E_1\} \\ &\quad \cup \{([x, y_1], [x, y_2] \mid (y_1, y_2) \in E_2\}. \end{aligned}$$

Ortogonální sítě - *orthogonal networks*

Příklad: Kartézský součin dvou grafů



Ortogonální sítě - *orthogonal networks*

Existují dva typy základních ortogonálních sítí:

- lineární řetězec - *linear array*
- kruh - *ring*.

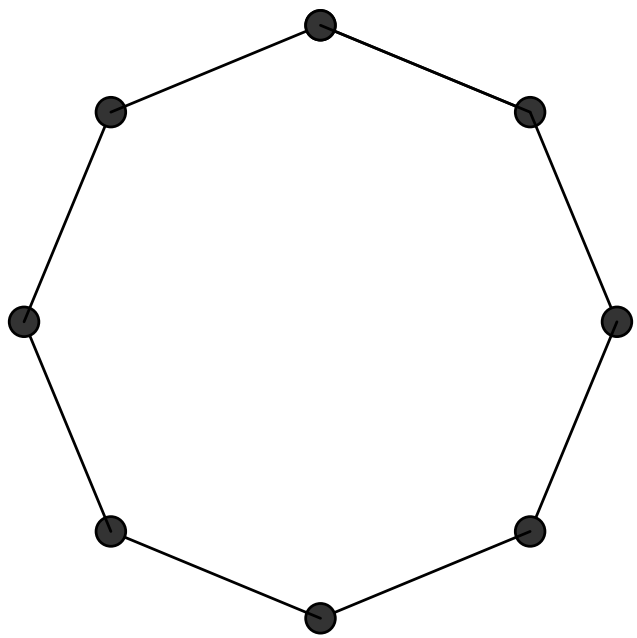
Ostatní sítě se odvozují pomocí kartézských součinů grafů.

Lineární řetězec - *linear array*

Lineární řetězec je graf v němž všechny vrcholy kromě prvního a posledního mají dva sousedy.

Kruh - *ring*

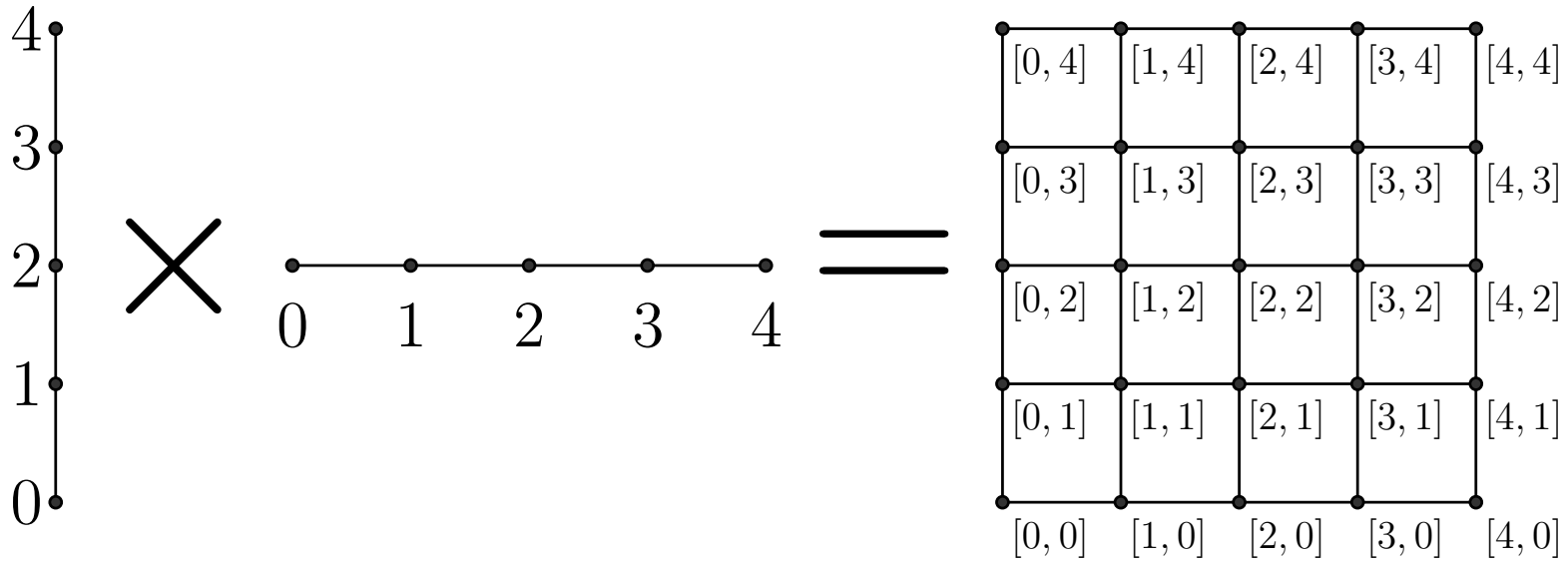
Kruh je graf, který odpovídá lineárnímu řetězci v němž je spojen první a poslední vrchol.



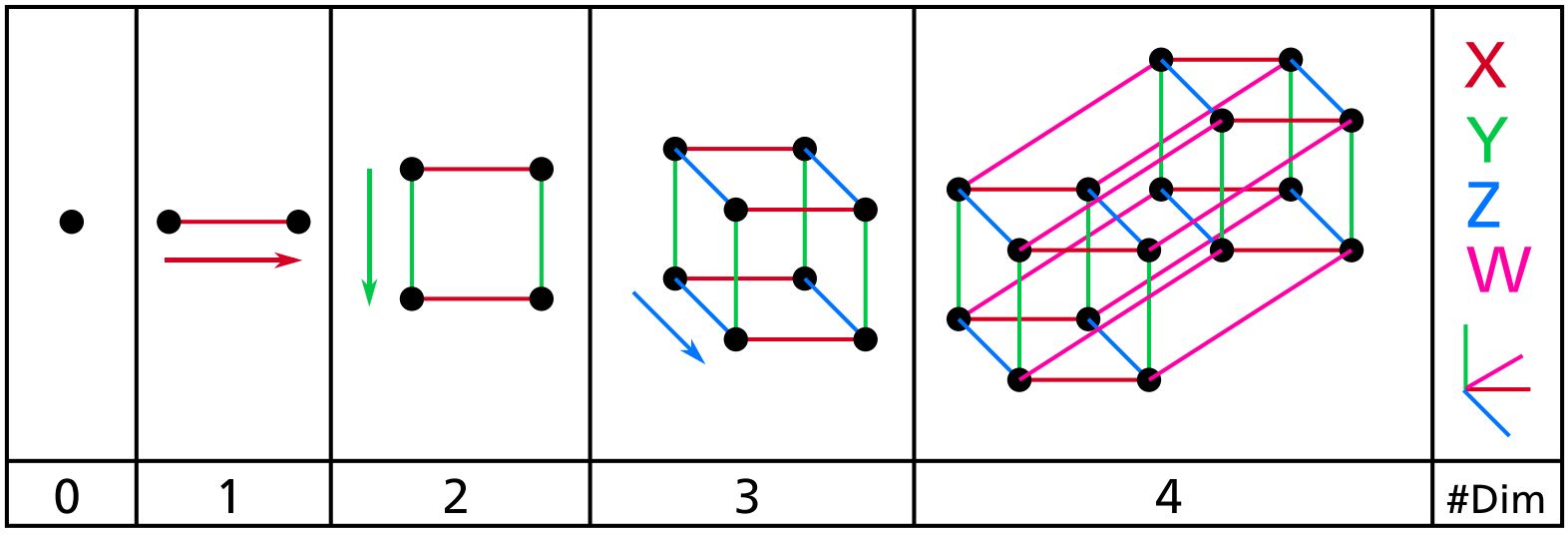
n-rozměrná síť - *mesh*

- je definována jako kartézský součin *n* lineárních řetězců obecně různé délky
- mají-li všechny řetězce v kartézském součinu dva vrcholy, dostaneme hyperkrychli - *hypercube* - Q_n

n-rozměrná síť - *mesh*



Hyperkrychle - *hypercube*



Zdroj: Wikipedie

n-rozměrný torus

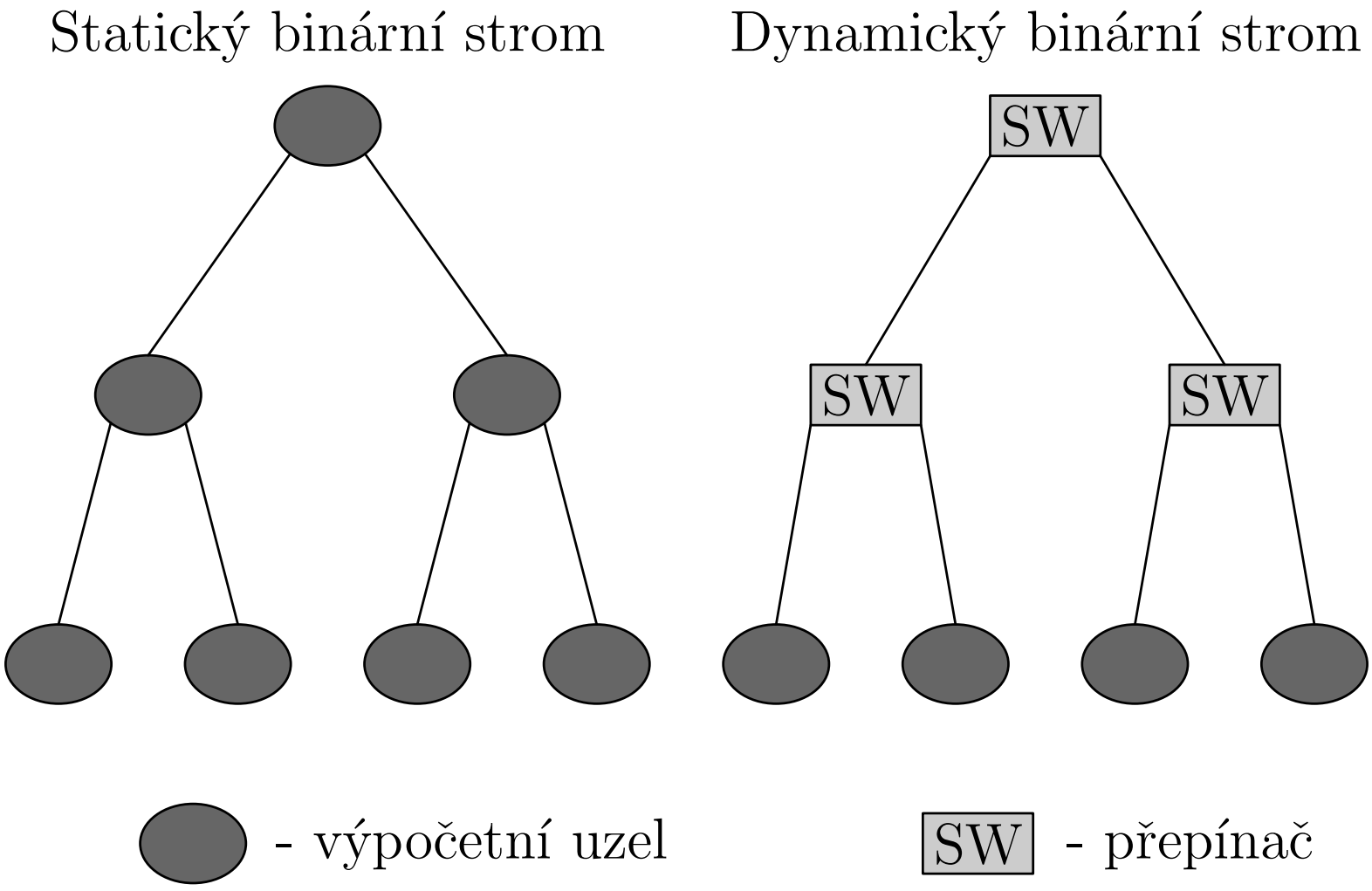
- je definován jako kartézský součin n kruhů obecně různé délky

Poznámka: Smíšené kartézské součiny lineárních řetězců a kruhů se neuvažují.

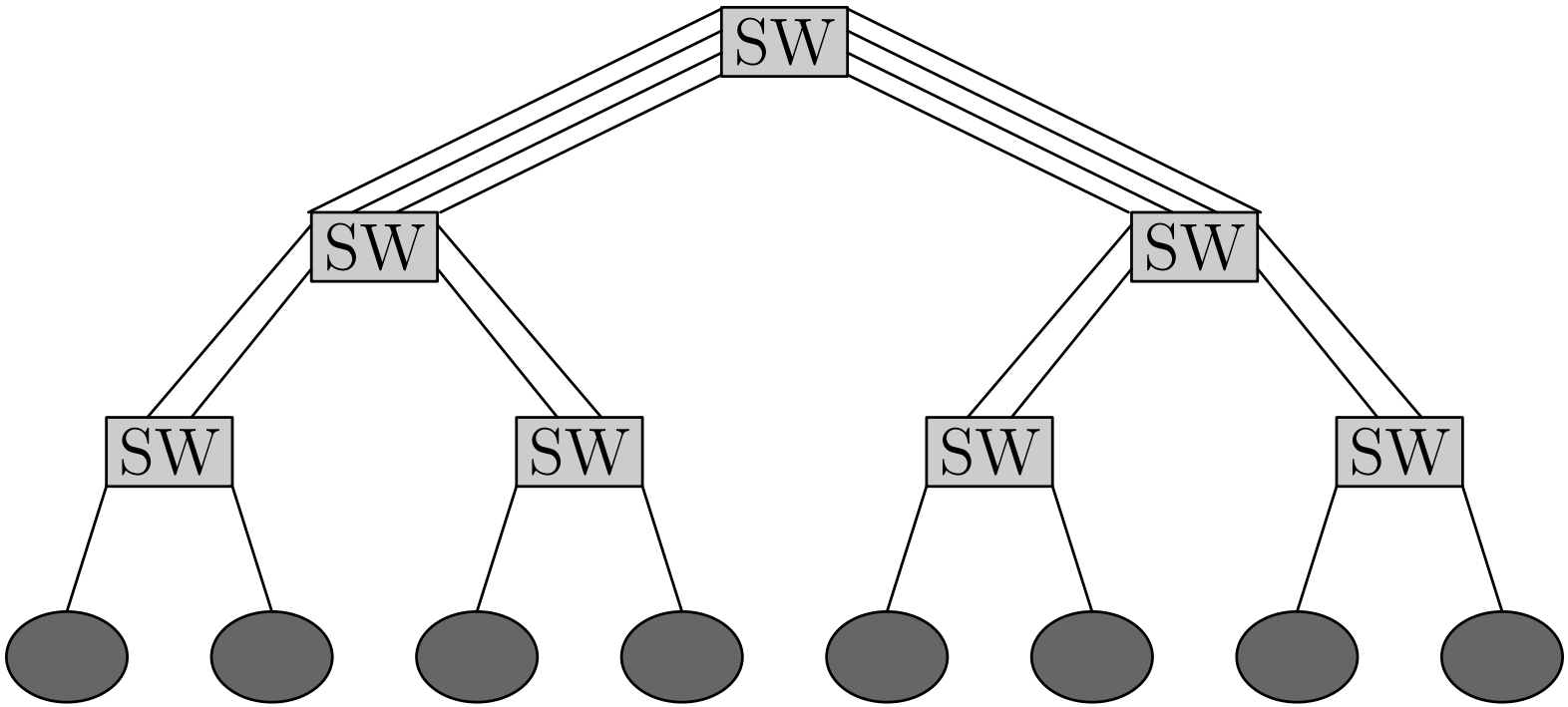
Stromové sítě - *tree-based networks*

- stromové sítě jsou popsány grafem, pro který platí, že mezi libovolným párem uzlů existuje jen jedna cesta
- jde například o
  - lineární řetězec
  - hvězdu
  - **binární strom**
- binární stromy se dělí na statické a dynamické

Stromové sítě - *tree-based networks*



Tlustý strom - *fat tree*



Složitost komunikace v komunikačních sítích I.

Nyní nás bude zajímat doba, za kterou zpráva (určitý objem dat) dojde po síti od zdroje k cíli.
Tato doba závisí na následujících parametrech:

- **inicializace** (*startup time*) - t_s
  - zahrnuje čas nutný ke zpracování zprávy u odesílatele a příjemce
  - skládá se z
    - vytvoření hlavičky
    - napočítání korekčních informací
    - provedení routovacícho algoritmu
    - nastavení síťového rozhraní
  - vyskytuje se pouze jednou při posílání jedné zprávy

Složitost komunikace v komunikačních sítích II.

- **prodleva na routeru** (*per-hop time, node latency*) - t_h
  - je to čas potřebný k určení správného výstupního kanálu/bufferu na routeru
  - to následně určuje, jakým směrem bude zpráva dále poslána
- **přenos** (*per-word time*) - t_w
  - je to doba potřebná k přenosu celé zprávy od jednoho uzlu k dalšímu
  - pokud je přenosová rychlost kanálu r slov za sekundu, pak $t_w = 1/r$

Circuit routing I.

1. Circuit routing

- přenos začíná vytvořením obvodu od zdroje k cíli = alokace celé cesty
  - vyšle se malá sonda, která hledá cestu k cíli
  - sonda po cestě alokuje spoje pro následné poslání zprávy
  - když sonda dosáhne cíle, pošle zpět potvrzující signál
- po nalezení celé cesty probíhá přenos zprávy
  - zpráva se nedělí na pakety [1]
  - během přenosu zprávy nejsou využívány žádné vyrovnávací paměti (buffery)
  - obvod je udržován po celou dobu přenosu
- po ukončení přenosu je obvod uvolněn

[1] Paket značí blok přenášených informací počítačovou sítí - http://en.wikipedia.org/wiki/Packet_(information_technology)

Circuit routing II.

Časová náročnost:
Je-li p velikost sondy, l délka cesty od zdroje k cíli a m délka zprávy, pak trvá

- nalezení cíle - $l(t_h + pt_w)$
- odeslání potvrzujícího signálu - lt_w
  - potvrzující signálu má délku 1 slovo
- vlastní přenos dat - mt_w
  - data jsou přenášena jakoby po jednom dlouhém datovém spoji.

Celkem dostáváme:

$$t_{CR} = t_s + l[t_h + (p+1)t_w] + mt_w$$

Circuit routing III.

- tento způsob je vhodný pro přenášení dlouhých zpráv s malou četností
- vztah pro t_{CR} neobsahuje člen tvaru lm, tj. téměř nezáleží na vzdálenosti komunikujících uzlů
- má-li se současně přenášet hodně zpráv, může se stát, že cesty dvou zpráv se budou "křížit"

Store-and-forward routing I.

2. Store-and-forward routing

- jde o pravý opak *circuit routing*
- zpráva putuje od jednoho uzlu ke druhému
- vždy je celá načtena do vyrovnávací paměti routeru a až poté je posílána dál

Store-and-forward routing II.

Časová náročnost:
Je-li l délka cesty od zdroje k cíli a m délka zprávy, pak trvá

- přenos celé zprávy - $l(t_h + mt_w)$.

Celkem dostáváme:

$$t_{SFR} = t_s + l(t_h + mt_w)$$

Store-and-forward routing II.

Časová náročnost:
Je-li l délka cesty od zdroje k cíli a m délka zprávy, pak trvá

- přenos celé zprávy - $l(t_h + mt_w)$.

Celkem dostáváme:

$$t_{SFR} = t_s + l(t_h + mt_w)$$

- tento způsob není vhodný pro posílání velkých zpráv na delší vzdálenosti
  - časová náročnost obsahuje výraz tvaru lm
- neblokuje tolik linky jako *circuit routing*
- toto směrování je vhodné hlavně pro krátké zprávy s velkou četností

Store-and-forward routing II.

Časová náročnost:
Je-li l délka cesty od zdroje k cíli a m délka zprávy, pak trvá

- přenos celé zprávy - $l(t_h + mt_w)$.

Celkem dostáváme:

$$t_{SFR} = t_s + l(t_h + mt_w)$$

- tento způsob není vhodný pro posílání velkých zpráv na delší vzdálenosti
  - časová náročnost obsahuje výraz tvaru lm
- neblokuje tolik linky jako *circuit routing*
- toto směrování je vhodné hlavně pro krátké zprávy s velkou četností

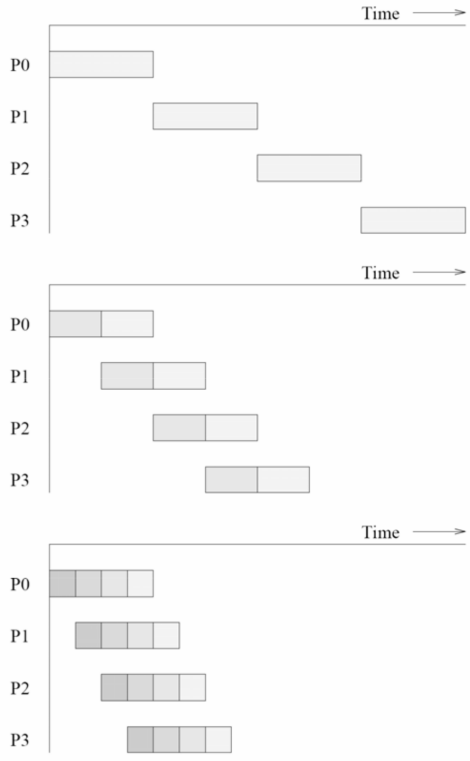
Poznámka: Nyní se budeme snažit nějaký kompromis mezi *circuit routing* a *store-and-forward routing*.

Packet routing I.

3. Packet routing

- využijeme možnosti rozdělit zprávu na menší celky tj. na pakety
- následně využijeme principu *pipelineningu*
  - každý uzel čeká jen na načtení celého paketu a ten ihned posílá dál
  - jakmile tedy druhý uzel přijal první paket, může ho odesílat třetímu uzlu a současně přijímat druhý paket od prvního uzlu

Packet routing I.



(a) A single message sent over a store-and-forward network

(b) The same message broken into two parts and sent over the network.

(c) The same message broken into four parts and sent over the network.

$$t_{comm} = t_s + (mt_w + t_h)l.$$

$$t_{comm} = t_s + mlt_w.$$

$$t_{comm} = t_s + lt_h + t_w m.$$

Zdroj: A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing

`http://www-users.cs.umn.edu/~karypis/parbook/`

Packet routing II.

Časová náročnost:

Je-li

- l délka cesty od zdroje k cíli
- m délka zprávy
- velikost paketu $r + s$ slov
  - r odpovídá vlastním datům
  - s je velikost hlavičky a korekčních informací,

pak trvá

- rozdělení zprávy na pakety (je úměrné velikosti zprávy) - mt_p
  - t_p je čas nutný k převedení jednoho slova zprávy do paketové formy
- pro jednoduchost předpokládáme, že všechny pakety jdou stejnou cestou
  - to nemusí být vždy pravda a nemusí to být vždy výhodné
- první paket dosáhne cíl za čas $lt_h + (r + s)t_w$
- další paket dosáhne cíl po $(r + s)t_w$
- po prvním paketu takových paketů dojde ještě $m/r - 1$

Packet routing III.

Celkem dostáváme

$$\begin{aligned} t_{PR} &= t_s + mt_p + lt_h + (r+s)\,t_w + \left(\frac{m}{r} - 1\right)(r+s)\,t_w \\ &= t_s + mt_p + lt_h + \left(\frac{m}{r}\right)(r+s)\,t_w \\ &= t_s + mt_p + lt_h + m\left(1 + \frac{s}{r}\right)t_w \\ &= t_s + lt_h + mt_z, \end{aligned}$$

kde $t_z = t_p + \left(1 + \frac{s}{r}\right)t_w$.

Packet routing IV.

- díky aplikaci *pipeliningu* jsme se opět zbavili součinu *lm* ve výrazu pro časovou náročnost
- zahodíme-li předpoklad, že všechny pakety musí jít stejnou cestou, pak nehrozí, že by dlouhá zpráva prakticky zablokovala všechny spoje na této cestě
- *packet routing* je vhodný pro sítě s vyšším obsahem chyb
- pro spolehlivější sítě můžeme použít ještě efektivnější směrování

Zjednodušený model složitosti komunikace I.

Routing	Časová náročnost
circuit routing	$t_{CR} = t_s + l[t_h + (p+1)t_w] + mt_w$
store-and-forward routing	$t_{SFR} = t_s + l(t_h + mt_w)$
packet routing	$t_{PR} = t_s + lt_h + mt_z$ pro $t_z = t_p + \left(1 + \frac{s}{r}\right)$

S výjimkou *store-and-forward routing* mají všechny modely přibližný tvar

$$t_{COMM} \approx t_s + lt_h + mt_w.$$

Zjednodušený model složitosti komunikace II.

Jak mimimalizovat čas nutný ke komunikaci ?

1. Provádět komunikaci ve větších celcích
   - **místo dvou krátkých zpráv, pošleme jednu delší**
   - jde o minimalizaci příspěvku členu t_s
   - u současných architektur platí $t_h << t_s$ a $t_w << t_s$
   - !!! tato optimalizace se opravdu vyplatí !!!
2. Minimalizace objemu dat
   - jde o minimalizaci příspěvku členu mt_w
   - pokud je možné redukovat délky posílaných zpráv, pak je to také užitečná optimalizace

Zjednodušený model složitosti komunikace III.

3. Provádět komunikaci na kratší vzdálenosti
   - jde o minimalizaci příspěvku členu mt_w
   - programátor ale vetšinou nemůže ovlivnit mapování procesů na procesory
     - některé funkce standardu MPI tomu ale mohou napomoci
   - některé architektury ale používají náhodné směrování dat, aby zabránily přetížení některých uzlů
   - jedná se o optimalizaci silně závislou na architektuře
   - platí, že $t_h << t_s$, a protože i $l << m$, pak je $lt_h << mt_w$
   - tato optimalizace se složitě provádí a nemá velký smysl

Zjednodušený model složitosti komunikace IV.

Protože platí $lt_h << mt_w$, můžeme komunikační model zjednodušit na

$$t_{COMM} \approx t_s + mt_w.$$

- tento model nezávisí na vzdálenosti
- je to, jako kdybychom měli k dispozici úplnou síť
- proto není nutné vyvíjet zvláštní programy pro různé topologie
- to však nemusí platit o sítích, které jsou náchylné k zahlcení
  - např. hvězda, lineární řetězec, kruh

Symetrické multipočítače IV.

Příkladem multipočítačů jsou zejména klastry (*cluster*).
Jaké jsou rozdíly mezi klastrem a počítačovou sítí?

- uzly klastru nemají připojené displeje
- na všech uzlech by měl běžet zcela identický operační systém
- je kladen větší důraz na kvalitu síťového spojení
  - gigabitový Ethernet (1Gb/sec, 100μsec), Myrinet (20Gb/sec, 7μsec)

Supepočítače

- dalším příkladem může být tzv. **massive parallel processing**
- jde o klastr se speciální vysoce rychlostní komunikační sítí
- typickými zástupci jsou superpočítače

Supepočítače

Cray Titan – Oak Ridge National Laboratory, 2012

- obsahuje 16 688 uzlů vybavených
  - AMD Opteron 6274 s 16 jádry
  - Nvidia Tesla K20X
- celkem 696 TB RAM
- 40 PB uložiště se souborovým systémem Lustre, 1.4 TB/s
- celkový výkon
  - 27 PFLOPS peak performance
  - 17.6 PFLOPS LINPACK

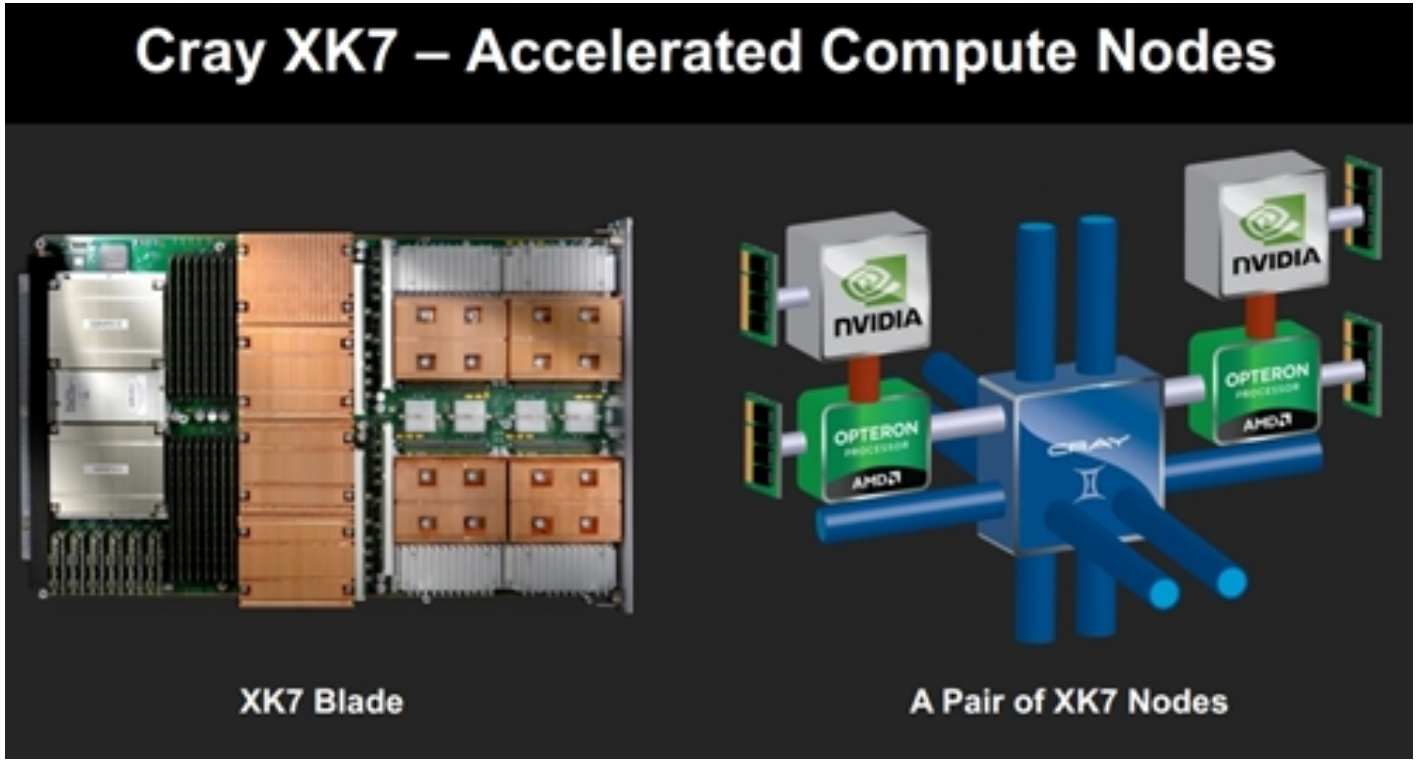
Supepočítače

Kabinet superpočítače Mira

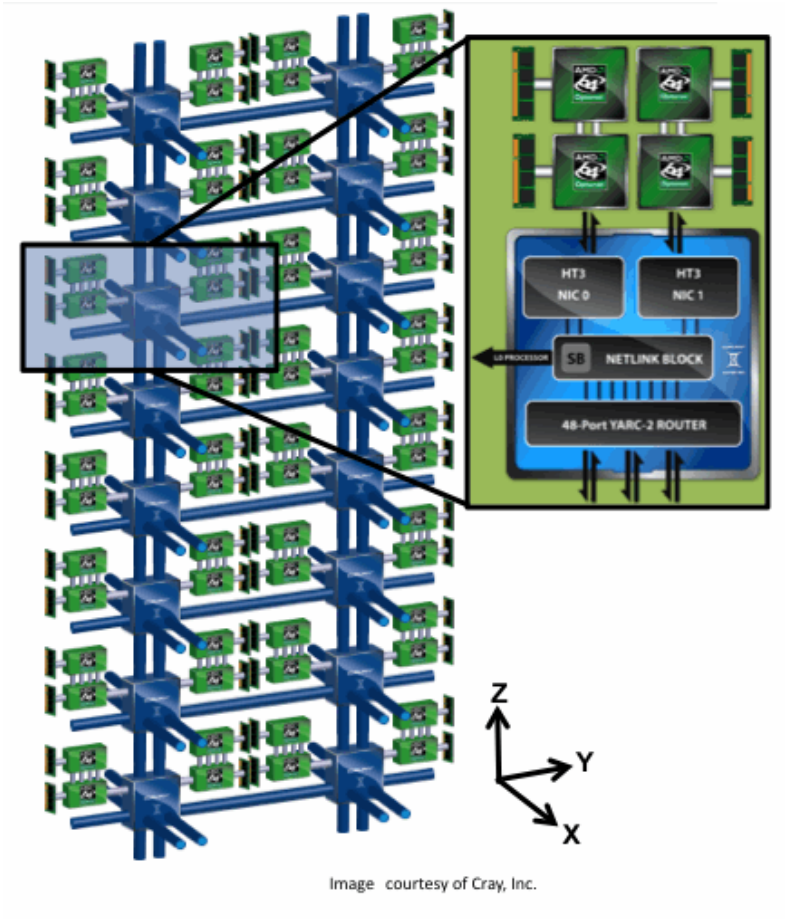
Supepočítače

- celý počítač je postaven na platformě Cray XK7
- ta je založena na tzv. kabinetech
- každý kabinet obsahuje 24 blade modulů
- každý modul obsahuje 4 uzly s jedním CPU a jedním GPU
  - CPU = 16 jádrový AMD Opteron 6200 + až 32 GB RAM
  - GPU = Nvidia Tesla K20 + až 6 GB RAM
- komunikaci zajišťuje systém Gemini Interconnect
- jeden čip pro dva uzly přenáší až 160 GB/s
- příkon jednoho kabinetu je až 54 kW

Supepočítače

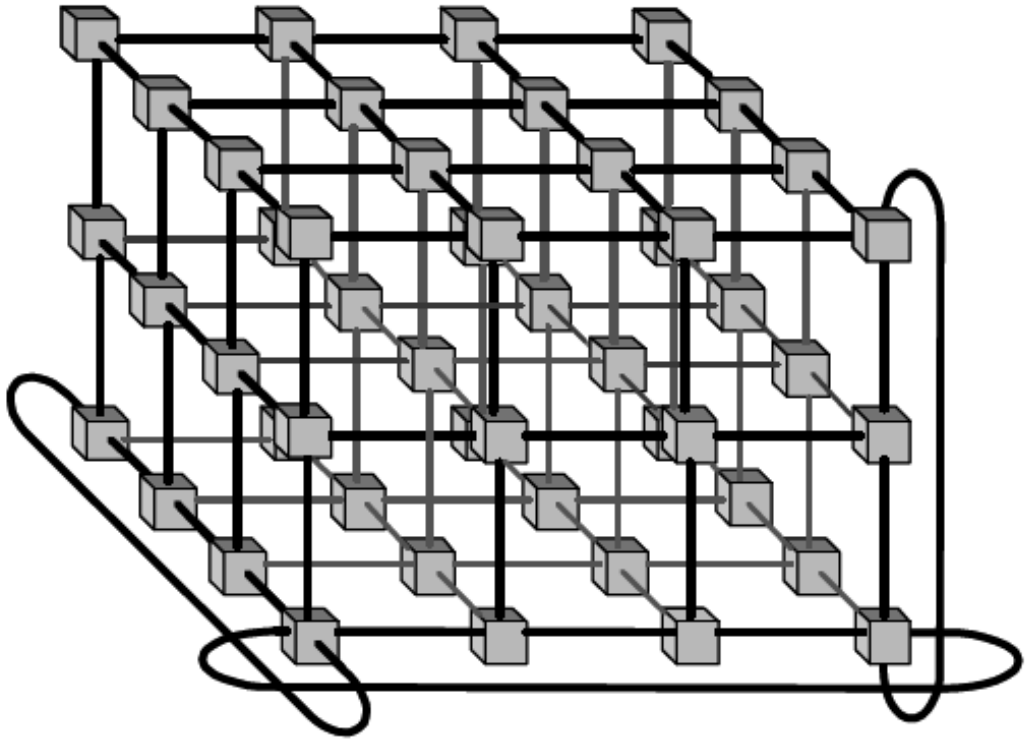


Supepočítače



Gemini interconnection

Supepočítače



Gemini interconnection

Supepočítače

- v současnosti Cray nabízí kabinety Cray XC30 a XC40
- XC40 lze osadit procesory Intel Xeon Haswell a kartami Intal Xeon Phi a Nvidia Tesla K40 (K80)
- systém Gemini nahradil systém Cray Aries založený na fat-tree topologii

Supepočítače

Superpočítač Summit – Oak Ridge National Laboratory (2017)

- 3 400 uzlů s procesory IBM Power9 a GPU Nvidia Volta
- propojení CPU a GPU pomocí Nvidia NVLink – až 20x rychlejší než PCI Express 16x
- každý uzel bude obsahovat 500 GB DDR4 RAM a 800GB odkládací paměti
- propojení pomocí fat-tree Mellanox EDR InfiniBand

IBM Power8

- 4 GHz
- 12 jader, každé zpracuje až 8 vláken
- 96 MB eDRAM L3 cache
- 128 MB off-chip L4 cache