Úvod do distribuovaných výpočtů

B4B36PDV – Paralelní a distribuované výpočty

Osnova

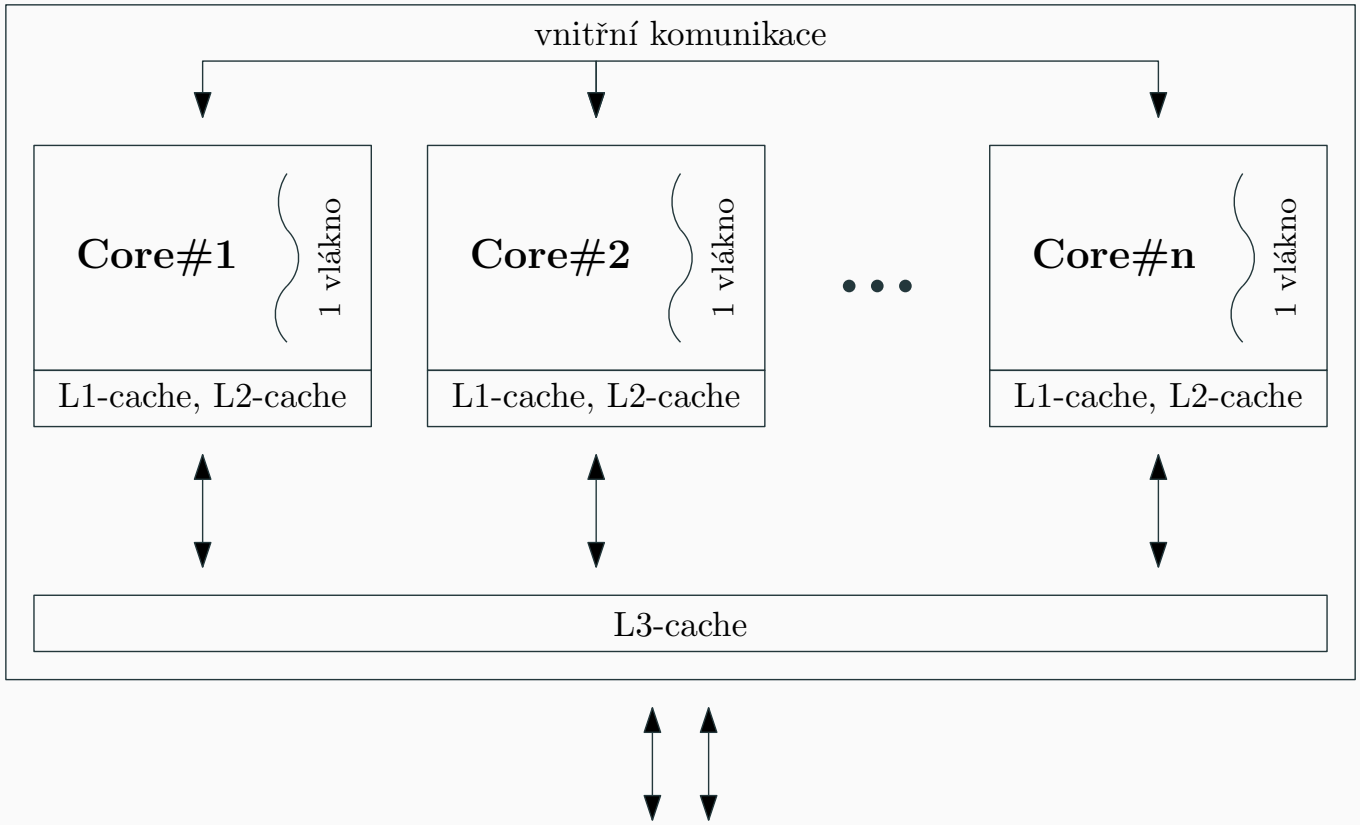
- Distribuovaný výpočet
- DSand framework
- Distribuované prohledávání

Vyžadujeme samostatnou práci na všech úlohách.

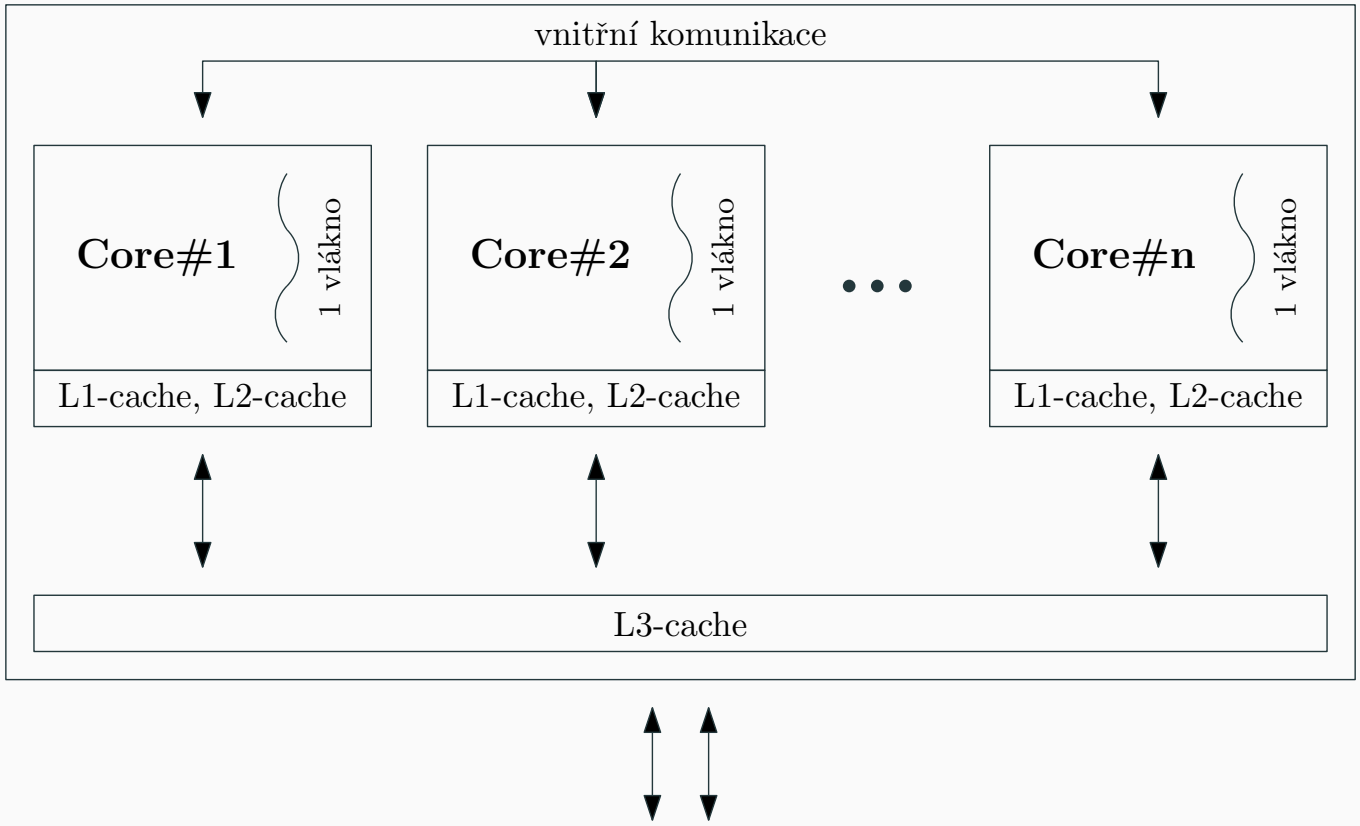
⚠ Plagiáty jsou zakázané. Nepřidělávejte prosím starosti nám, ani sobě.

Distribuovaný výpočet

Moderní procesor



Moderní procesor



Proč bychom mohli chtít tento model opustit?

Motivace distribuovaného výpočtu

Motivace distribuovaného výpočtu

- I ty nejšílenější procesory mají omezený počet vláken (za šílenou cenu)
  (Xeon Platinum 9282, 112 vláken, ∼$25000-$50000 ? (2018))

Motivace distribuovaného výpočtu

- I ty nejšílenější procesory mají omezený počet vláken (za šílenou cenu)
  (Xeon Platinum 9282, 112 vláken, ∼$25000-$50000 ? (2018))
- Přístupy do sdílené paměti jsou drahé
  (I pokud nepotřebujeme synchronizovat! – sběrnice má omezenou kapacitu)

Motivace distribuovaného výpočtu

- I ty nejšílenější procesory mají omezený počet vláken (za šílenou cenu)
  (Xeon Platinum 9282, 112 vláken, ∼$25000-$50000 ? (2018))
- Přístupy do sdílené paměti jsou drahé
  (I pokud nepotřebujeme synchronizovat! – sběrnice má omezenou kapacitu)

- Jediný „stroj“ je *single point of failure*
  (Pokud nám selže, tak jsme v háji)

Motivace distribuovaného výpočtu

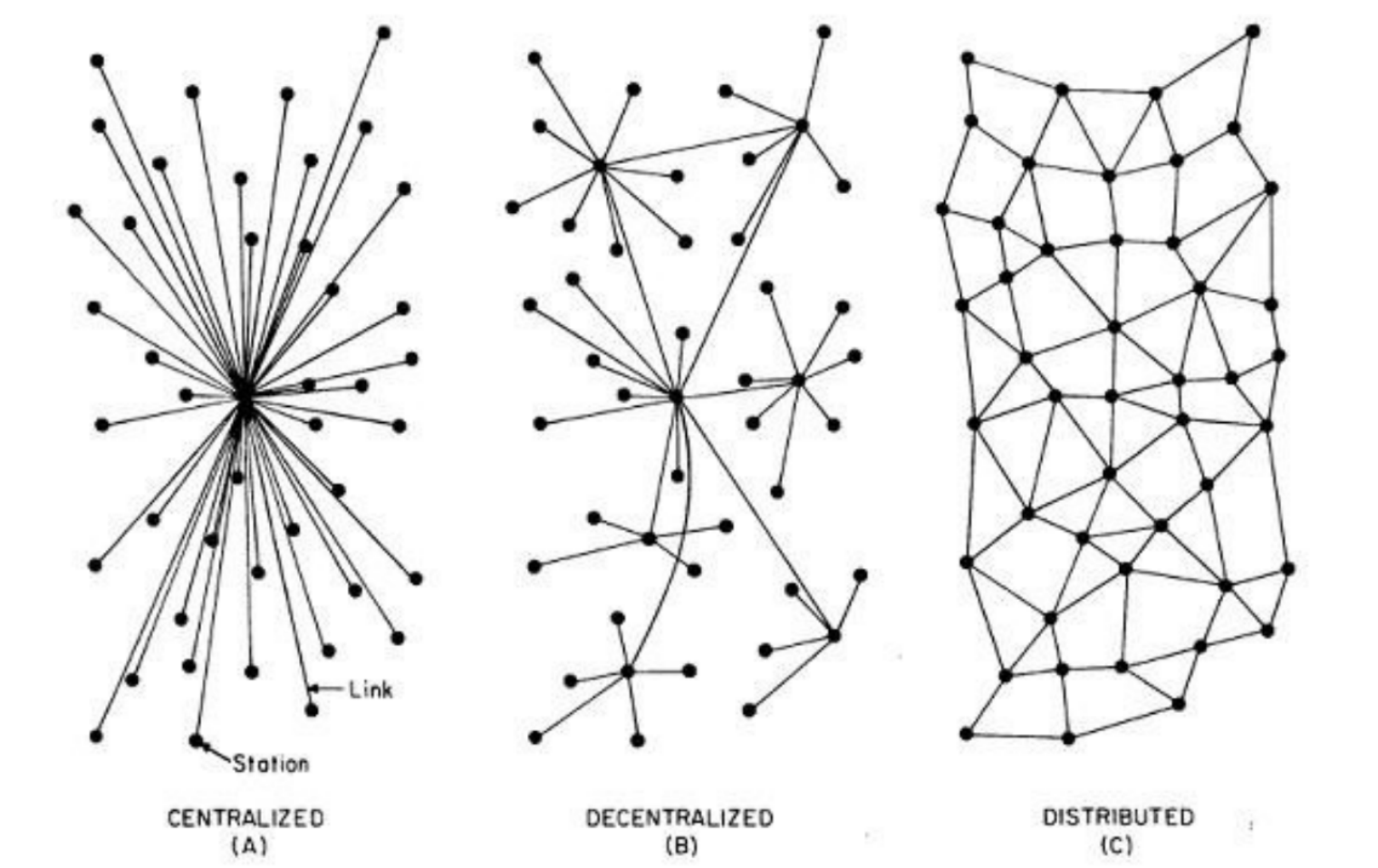
- I ty nejšílenější procesory mají omezený počet vláken (za šílenou cenu)
  (Xeon Platinum 9282, 112 vláken, ∼$25000-$50000 ? (2018))
- Přístupy do sdílené paměti jsou drahé
  (I pokud nepotřebujeme synchronizovat! – sběrnice má omezenou kapacitu)

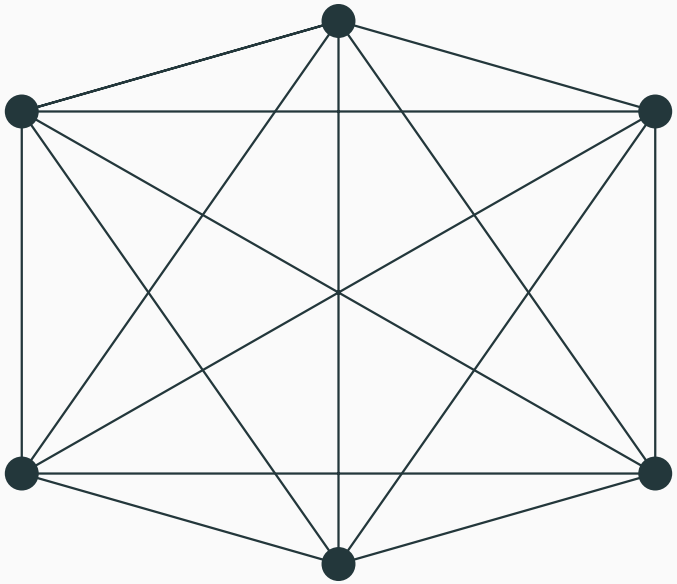
- Jediný „stroj“ je *single point of failure*
  (Pokud nám selže, tak jsme v háji)
- Chceme být geograficky blíže cílovým uživatelům
  (Nižší latence, ...)

Motivace distribuovaného výpočtu

- I ty nejšílenější procesory mají omezený počet vláken (za šílenou cenu)
  (Xeon Platinum 9282, 112 vláken, ∼$25000-$50000 ? (2018))
- Přístupy do sdílené paměti jsou drahé
  (I pokud nepotřebujeme synchronizovat! – sběrnice má omezenou kapacitu)

- Jediný „stroj“ je *single point of failure*
  (Pokud nám selže, tak jsme v háji)
- Chceme být geograficky blíže cílovým uživatelům
  (Nižší latence, ...)
- „Důvěryhodnost“ výpočtu
  (Pokud nám výpočet zverifikuje několik *nezávislých* strojů, je pravděpodobně správný)


Link
Station
CENTRALIZED
(A)
DECENTRALIZED
(B)
DISTRIBUTED
(C)



Na jaké problémy ale narazíme?

Na jaké problémy ale narazíme?

- Nespolehlivá komunikace (ztracené/zpožděné zprávy)
- Nespolehlivé výpočetní jednotky (musíme počítat se selháním procesu)

Na jaké problémy ale narazíme?

- Nespolehlivá komunikace (ztracené/zpožděné zprávy)
- Nespolehlivé výpočetní jednotky (musíme počítat se selháním procesu)

A v nejhorším případě až:

- Zprávy doručené v chybném pořadí (UDP vs. TCP)
- Poškozené zprávy
- Procesy, které se nás snaží cíleně podvést (například, infikované uzly)

Čím se budeme zabývat?

- Výpočet provádí současně více oddělených výpočetních uzlů
  (často i geograficky – my budeme ale distribuovanost simulovat)
- Cíle:
  - Zrychlit výpočet
  - Robustnost výpočtu
- (5 týdnů)

Úlohy z distribuované části (min. 9 bodů)

- 2 malé úlohy — 4 body
- Semestrální práce — 14 bodů

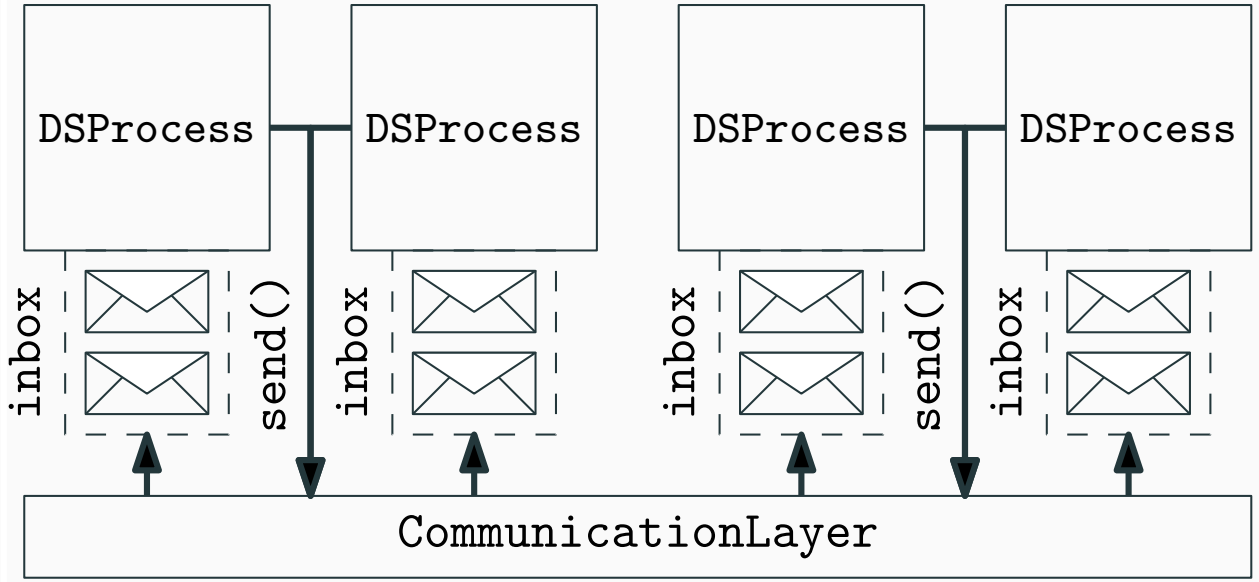
Čím se budeme zabývat?

- Výpočet provádí současně více oddělených výpočetních uzlů
  (často i geograficky – my budeme ale distribuovanost simulovat)
- Cíle:
  - Zrychlit výpočet
  - Robustnost výpočtu
- (5 týdnů)

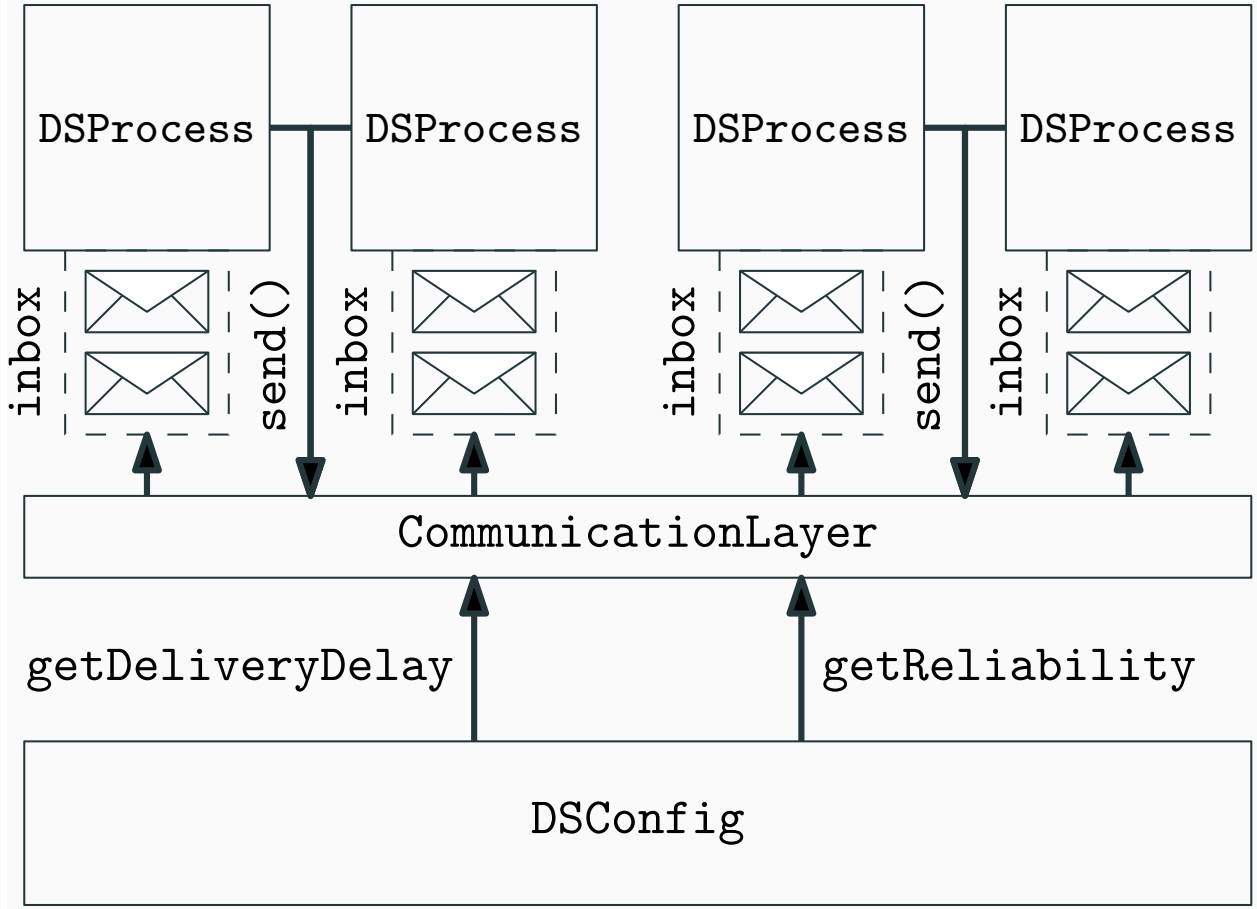
Úlohy z distribuované části (min. 9 bodů)

- 2 malé úlohy — 4 body
- Semestrální práce — 14 bodů

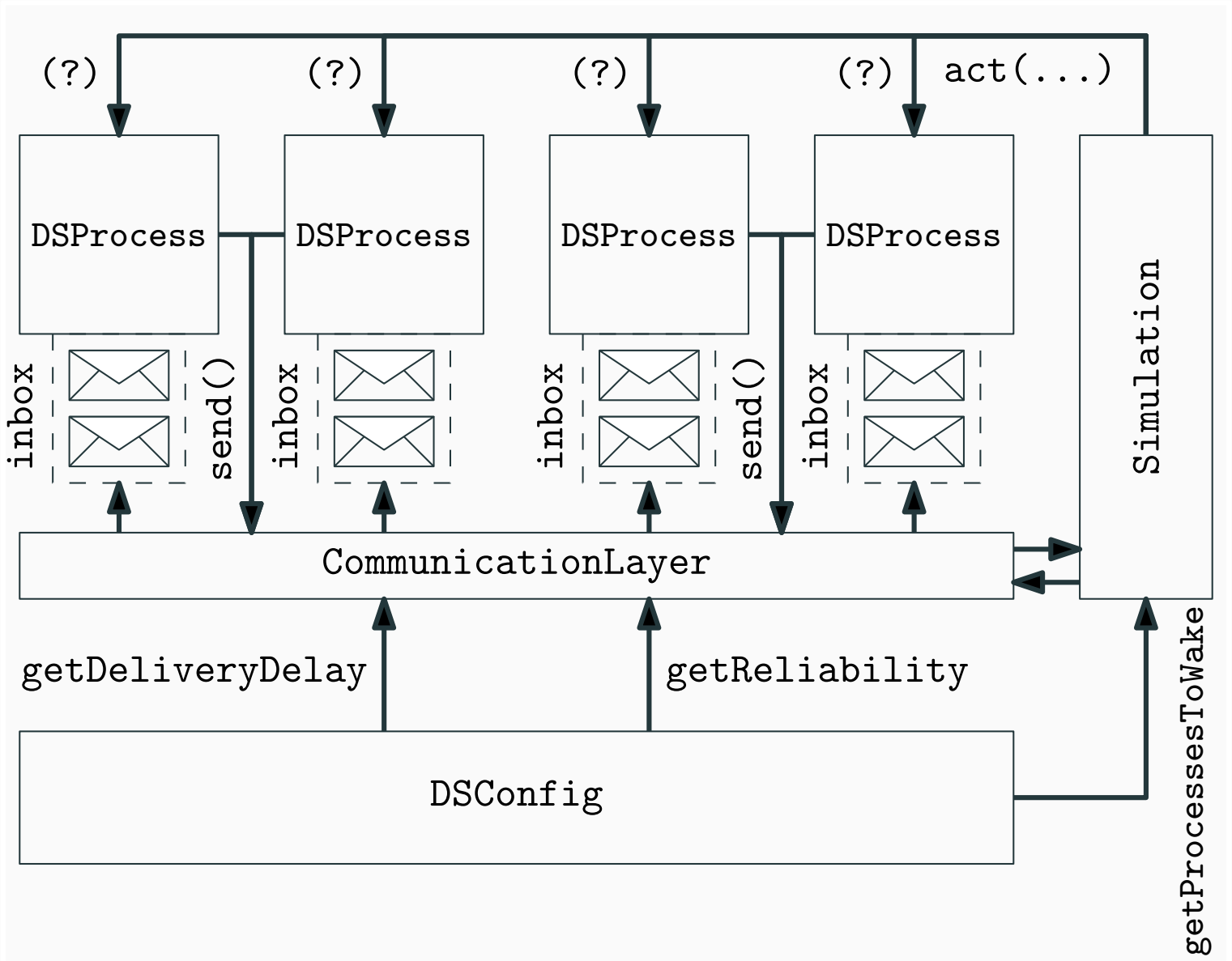
DSand framework



DSand framework



DSand framework



Jednoduchá výměna zpráv

Seznamte se s frameworkem DSand

Spusťte hlavní třídu `demo.Main` a vyzkoušejte si framework DSand. Můžete si i zkusit změnit parametry systému ve třídě `demo.DSConfig` a pozorovat, jaký vliv tato změna má na chování distribuovaného systému.

Synchronní distribuovaný systém

Synchronní distribuovaný systém

Uvažujme na začátek ideální distribuovaný systém...

- Zprávy se nám **neztrácí** a jsou doručované po konstantní době
- Procesy nehavarují a pracují **synchronně** (probouzí se všichni naráz)

Synchronní distribuovaný systém

Uvažujme na začátek ideální distribuovaný systém...

- Zprávy se nám **neztrácí** a jsou doručované po konstantní době
- Procesy nehavarují a pracují **synchronně** (probouzí se všichni naráz)

To je ještě optimističtější předpoklad,
než jsme měli v paralelizaci!

Synchronní prohledávání do šířky

Hledáme nejkratší cestu do cíle.

- Stavový prostor je silně souvislý
- Neznáme dopředu velikost ani strukturu prostoru

Synchronní prohledávání do šířky

Hledáme nejkratší cestu do cíle.

- Stavový prostor je silně souvislý
- Neznáme dopředu velikost ani strukturu prostoru

Jak namapujeme stavový prostor na procesy v síti?

Synchronní prohledávání do šířky

Hledáme nejkratší cestu do cíle.

- Stavový prostor je silně souvislý
- Neznáme dopředu velikost ani strukturu prostoru

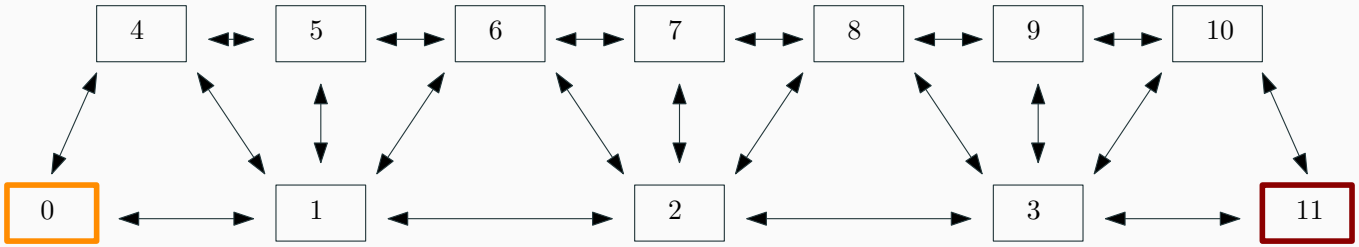
Jak namapujeme stavový prostor na procesy v síti?

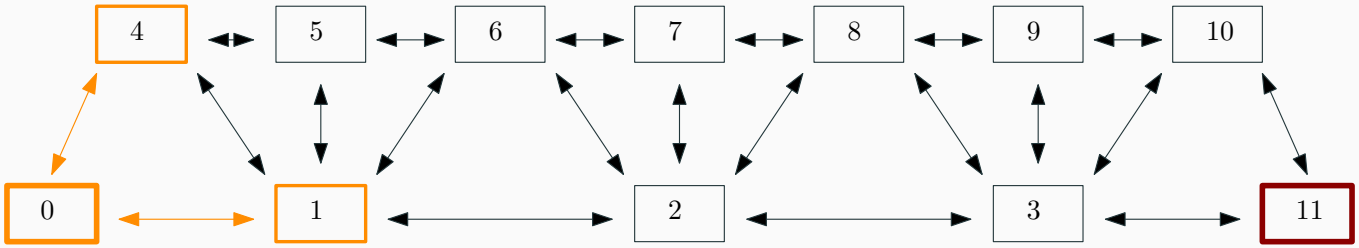
→ Každý stav je **samostaný proces**.

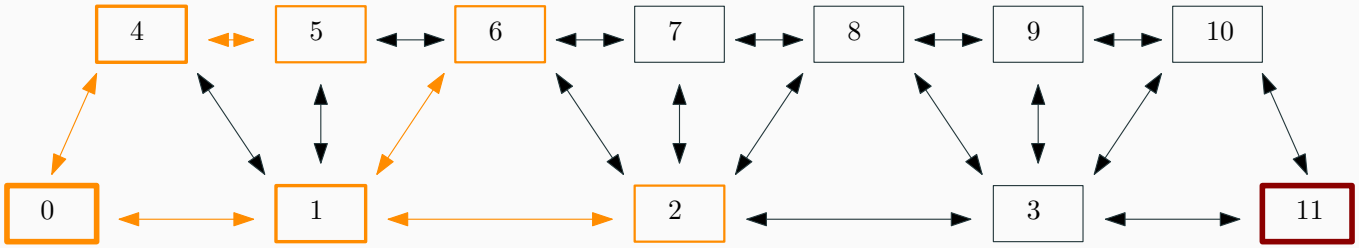
→ Každý proces zná jen své následovníky.

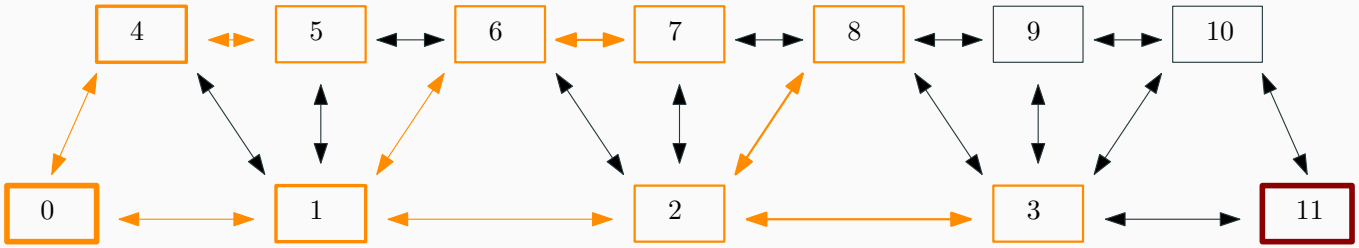
→ Jeden z procesů je **kořen**.

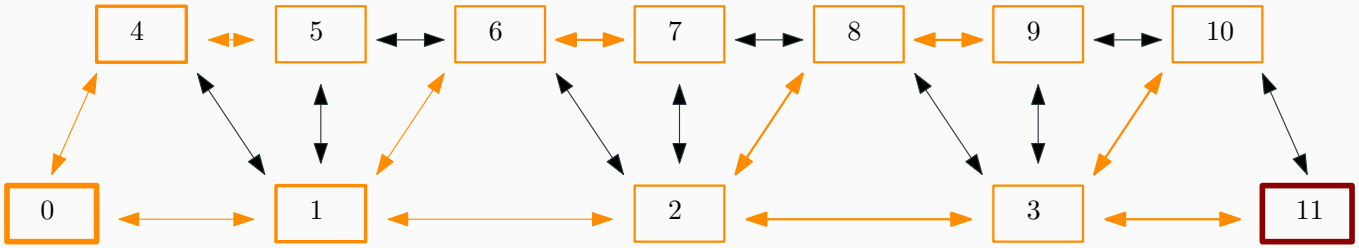
Kořen lze zvolit pomocí volby leadera!


4
5
6
7
8
9
10
0
1
2
3
11


4
5
6
7
8
9
10
0
1
2
3
11


4
5
6
7
8
9
10
0
1
2
3
11


4
5
6
7
8
9
10
0
1
2
3
11


4
5
6
7
8
9
10
0
1
2
3
11

Synchronní prohledávání do šířky

Jaký algoritmus bude provádět každý proces?

Synchronní prohledávání do šířky

Jaký algoritmus bude provádět každý proces?

Označujeme procesy, když jsou **poprvé navštíveny**

- Na začátku je označený jen kořen
- Kořen pošle zprávu `hledej` všem svým následovníkům

Synchronní prohledávání do šířky

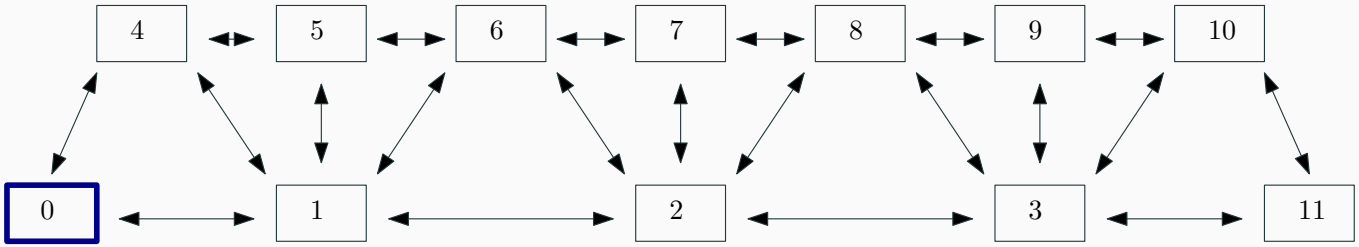
Jaký algoritmus bude provádět každý proces?

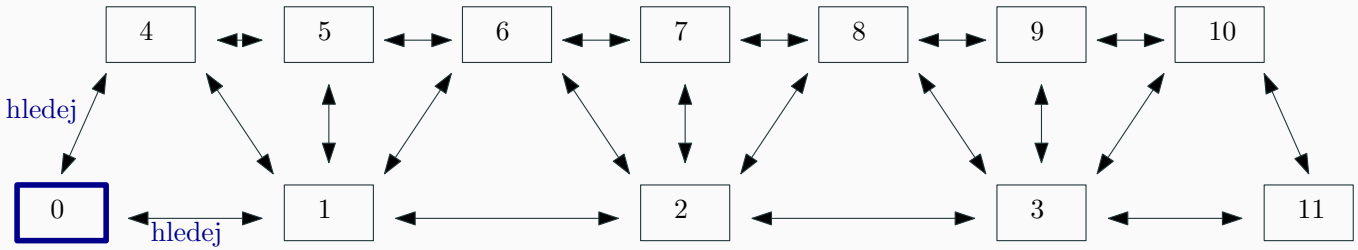
Označujeme procesy, když jsou **poprvé navštíveny**

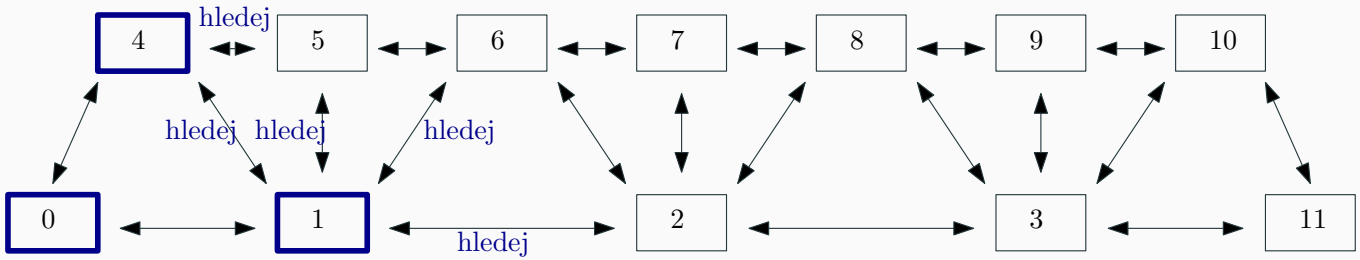
- Na začátku je označený jen kořen
- Kořen pošle zprávu `hledej` všem svým následovníkům

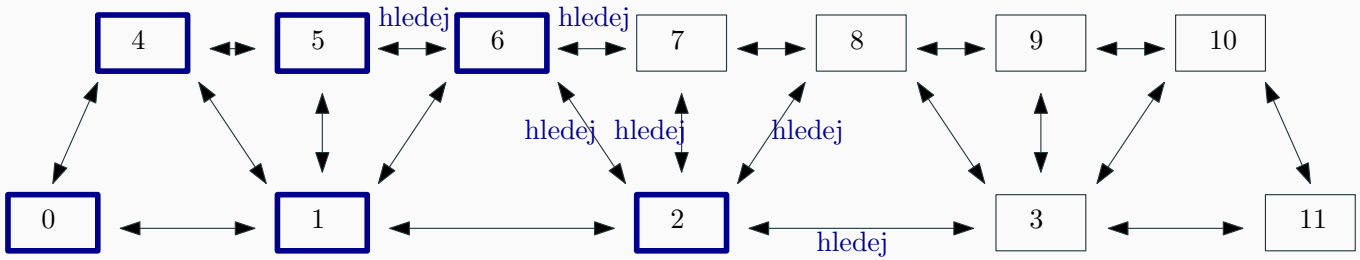
V každém kole *neoznačené* procesy, které přijmou zprávu `hledej`, provedou

- Označí se
- Určí jednoho z procesů, od kterého přijal zprávu, jako předka
- Pošlou zprávu `hledej` všem svým následovníkům


4
5
6
7
8
9
10
0
1
2
3
11


4
5
6
7
8
9
10
hledej
0
hledej
1
2
3
11


4
hledej
5
6
7
8
9
10
hledej
hledej
hledej
0
1
hledej
2
3
11


4
5
hledej
6
hledej
7
8
9
10
hledej
hledej
hledej
0
1
2
3
11
hledej

Synchronní prohledávání do šířky

Naimplementujte distribuované prohledávání do šířky

Do souboru `BFSProcess.java` doplňte kód procesů, které budou prohledávat síť do šířky. Předpokládejte, že systém je synchronní. Ve chvíli kdy najdete cíl, ukončete celý distribuovaný výpočet pomocí metody `terminateAll(...)`.

Jak vypsat posloupnost procesů na cestě k cíli?

Jak vypsat posloupnost procesů na cestě k cíli?

Propagujeme informaci napříč sítí!

Synchronní prohledávání do šířky s předáváním cesty

Doimplementujte konstrukci cesty z kořene do cíle

Přidejte do Vašeho algoritmu pro distribuované prohledávání zpětnou rekonstrukci cesty z cíle do kořenu. Ve chvíli kdy kořen bude znát celou cestu, vypište ji a ukončete distribuovaný výpočet.

Jak bude tento algoritmus fungovat, když systém nebude synchronní?

Jak bude tento algoritmus fungovat, když systém nebude synchronní?

Vyzkoušejte nesynchronní scénáře

Zkuste spustit scénáře `SlowOptimalPath.java`, ve kterém jsou zprávy pouze po optimální cestě zpožděny a `FailingNode2.java`, kdy se zprávy procesu 2 ztrácí. Jaké jsou výsledky jednotlivých scénářů?

Jak bude tento algoritmus fungovat, když systém nebude synchronní?

Vyzkoušejte nesynchronní scénáře

Zkuste spustit scénáře `SlowOptimalPath.java`, ve kterém jsou zprávy pouze po optimální cestě zpožděny a `FailingNode2.java`, kdy se zprávy procesu 2 ztrácí. Jaké jsou výsledky jednotlivých scénářů?

⚠ **Nemáme žádné garance.** Výsledek nemusí být optimální, dokonce ani nemusí nikdy doběhnout.

Díky za pozornost!

Budeme rádi za Vaši zpětnou vazbu! →

https://bit.ly/3b9oGzr