Vektorové hodiny a vzájemné vyloučení

B4B36PDV – Paralelní a distribuované výpočty

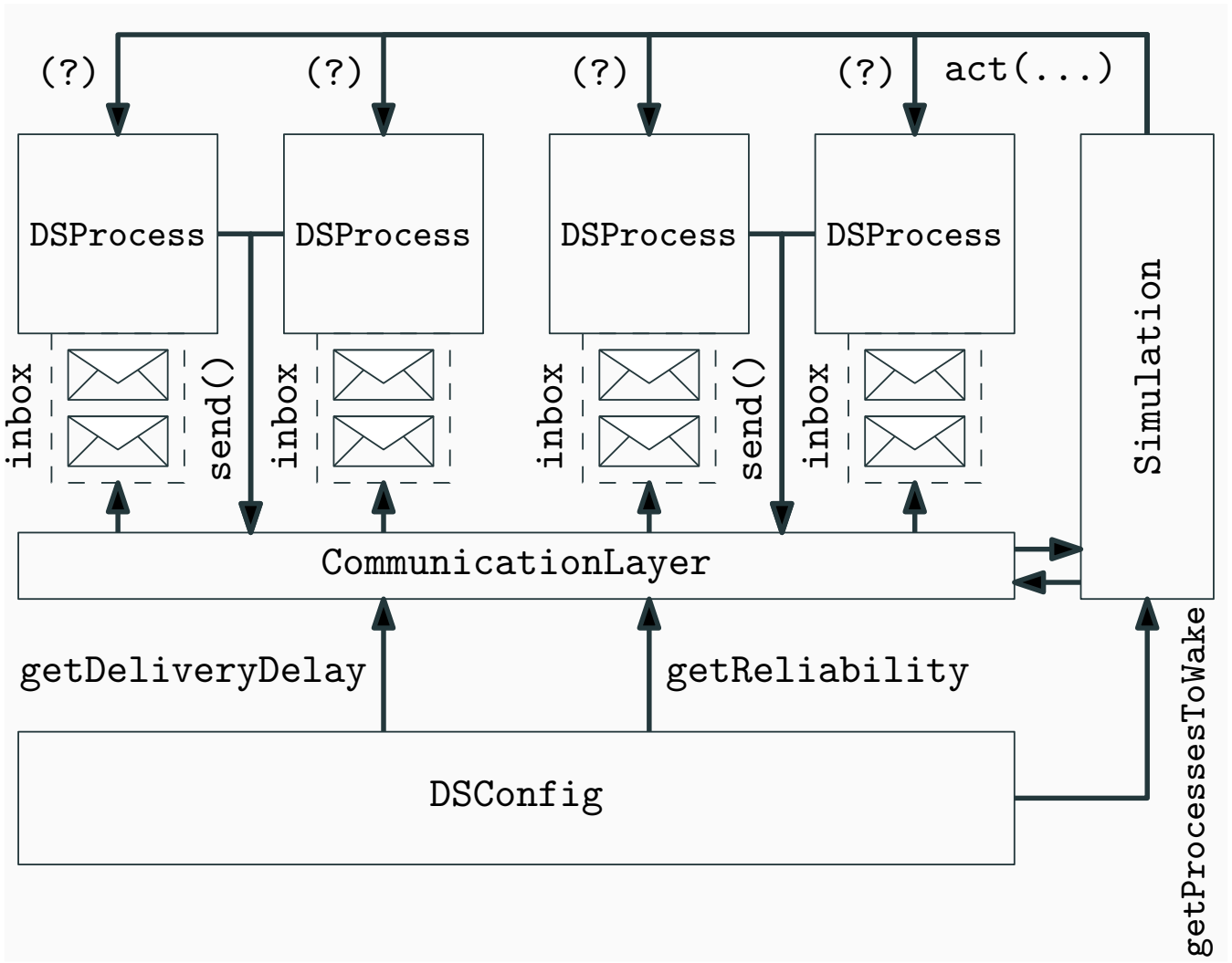
Osnova

- Opakování z minulého cvičení
- Vektorové hodiny
- Vzájemné vyloučení
- Zadání sedmé domácí úlohy

Opakování z minulého cvičení

http://goo.gl/a6BEMb

DSand framework



Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas
2. mohou sloužit k detekci porušení kauzality
3. informují příjemce zprávy o hodinách odesílatele
4. vynucují totální uspořádání událostí v systému
5. určují reálný čas, kdy byla zpráva poslána

Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas - FALSE
2. mohou sloužit k detekci porušení kauzality
3. informují příjemce zprávy o hodinách odesílatele
4. vynucují totální uspořádání událostí v systému
5. určují reálný čas, kdy byla zpráva poslána

Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas - FALSE
2. mohou sloužit k detekci porušení kauzality - TRUE
3. informují příjemce zprávy o hodinách odesílatele
4. vynucují totální uspořádání událostí v systému
5. určují reálný čas, kdy byla zpráva poslána

Např. pokudu komunikace dvou procesů na otázku T=5 přijde odpověď s T=3, je jasné že je to odpověď na jinou otázku. Pro více procesů ale tato situace nemusí být zdetekována.

Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas - FALSE
2. mohou sloužit k detekci porušení kauzality - TRUE
3. informují příjemce zprávy o hodinách odesílatele - TRUE
4. vynucují totální uspořádání událostí v systému
5. určují reálný čas, kdy byla zpráva poslána

Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas - FALSE
2. mohou sloužit k detekci porušení kauzality - TRUE
3. informují příjemce zprávy o hodinách odesílatele - TRUE
4. vynucují totální uspořádání událostí v systému - FALSE
5. určují reálný čas, kdy byla zpráva poslána

Není to tak, např. zpráva 3 jednoho procesu může být odeslána před zprávou 1 od jiného procesu.

Jakou roli hrají v DS logické hodiny?

1. zajišťují, že všechny procesy mají stejný čas - FALSE
2. mohou sloužit k detekci porušení kauzality - TRUE
3. informují příjemce zprávy o hodinách odesílatele - TRUE
4. vynucují totální uspořádání událostí v systému - FALSE
5. určují reálný čas, kdy byla zpráva poslána - FALSE

Čas a uspořádání událostí v DS (2. část)

Lamportův algoritmus

1. Každý proces má svoje lokální logické hodiny
   `int logicalTime = 0`
2. Před každou významnou událostí (obzvlášť posláním zprávy!) si proces lokální čas posune
   `++logicalTime`
3. Každé zprávě přiřadíme časovou značku $\texttt{msg.}T = \texttt{logicalTime}$
   (Tím říkáme přijímajícímu procesu, ať si upraví svůj čas!)
4. Přijetí zprávy je následkem jejího odeslání – pak musí platit $T(e) < T(e')$
   Po přijetí zprávy `msg` si proto musíme zaktualizovat svůj `logicalTime`:

$$\texttt{logicalTime} = 1 + \max\{\texttt{logicalTime}, \texttt{msg.}T\}$$

Lamportův algoritmus

1. Každý proces má svoje lokální logické hodiny
   `int logicalTime = 0`
2. Před každou významnou událostí (obzvlášť posláním zprávy!) si proces lokální čas posune
   `++logicalTime`
3. Každé zprávě přiřadíme časovou značku $\texttt{msg.}T = \texttt{logicalTime}$
   (Tím říkáme přijímajícímu procesu, ať si upraví svůj čas!)
4. Přijetí zprávy je následkem jejího odeslání – pak musí platit $T(e) < T(e')$
   Po přijetí zprávy `msg` si proto musíme zaktualizovat svůj `logicalTime`:

$$\texttt{logicalTime} = 1 + \max\{\texttt{logicalTime}, \texttt{msg.}T\}$$

⚠ **Skalární hodiny jsou stavebním kamenem mnoha algoritmů v DS!**

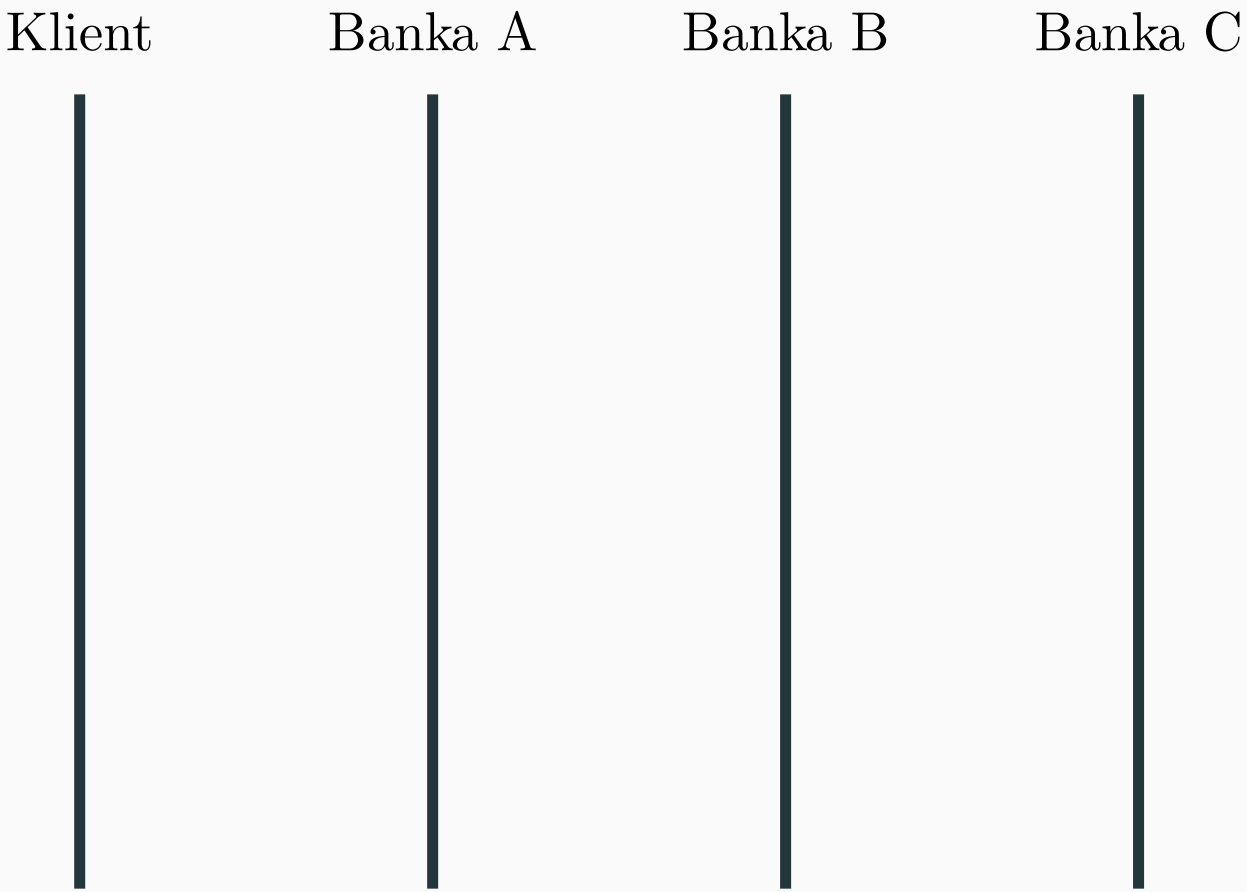
Chceme provést následující dvě operace v daném pořadí:

1. Převést všechny peníze z účtu v bance A na účet v bance B (`transfer_all(A, B)`)
2. Převést všechny peníze z účtu v bance B na účet v bance C (`transfer_all(B, C)`)

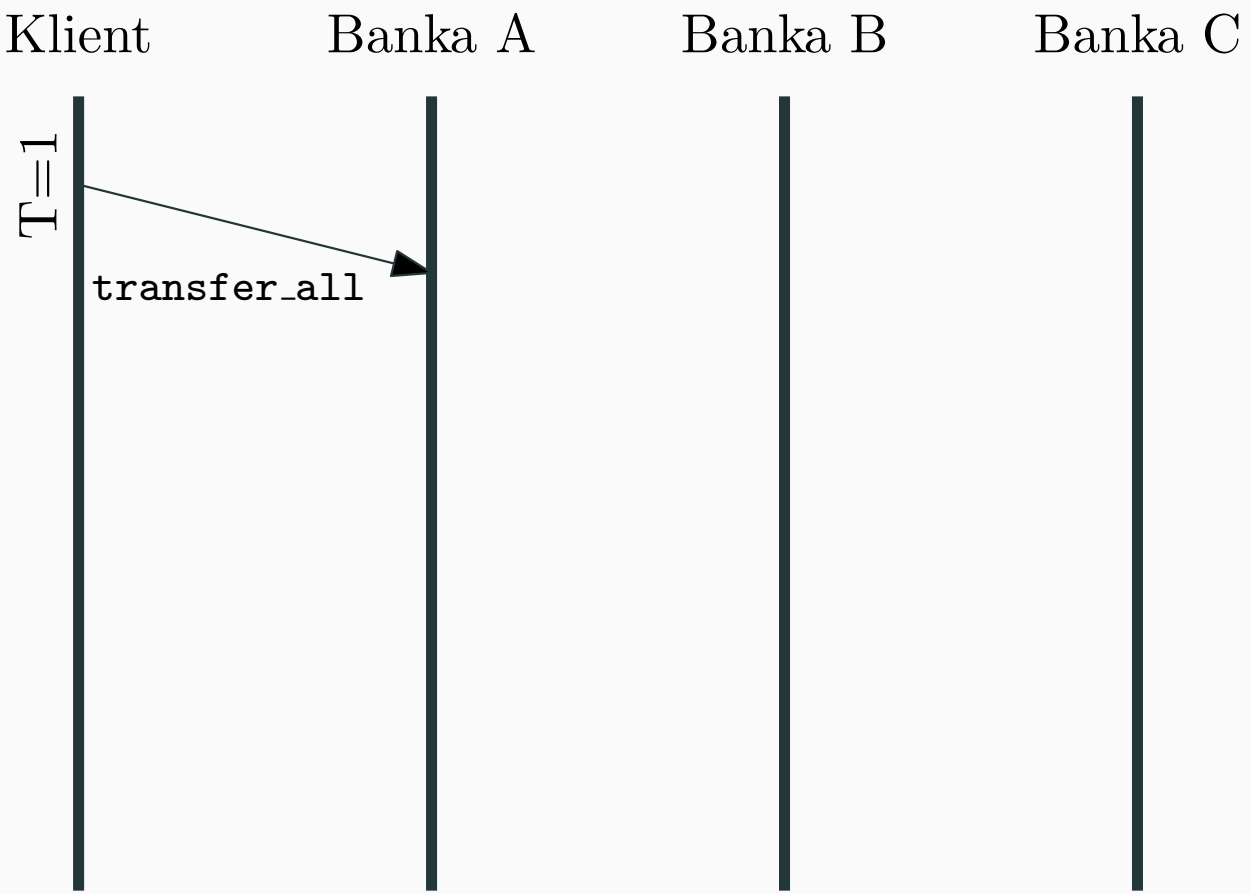
```
void transfer_all(int & from, int & to) {
        to += from;
        from = 0;
}

transfer_all(A, B);
transfer_all(B, C);
```

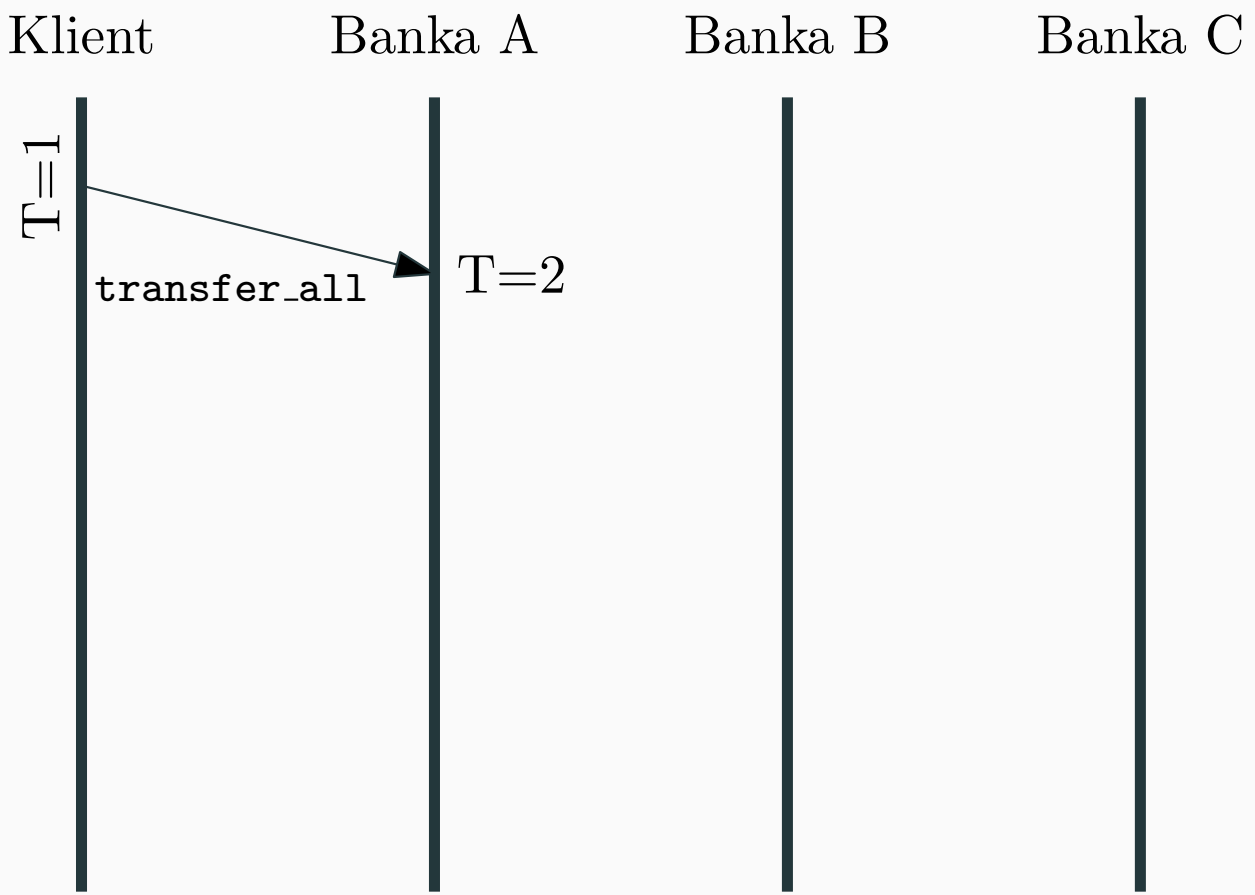
Jak to provést v distribuovaném systému?



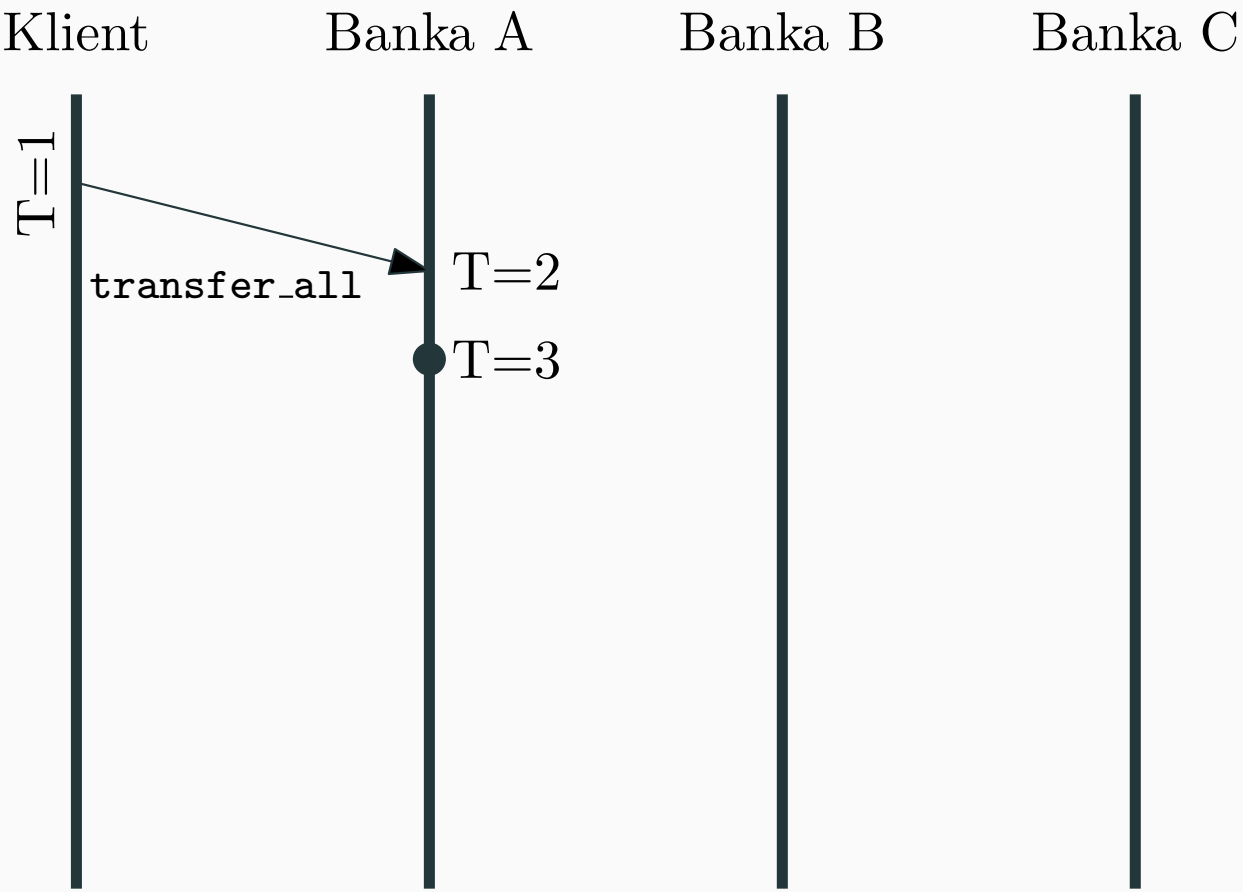
Jak to provést v distribuovaném systému?



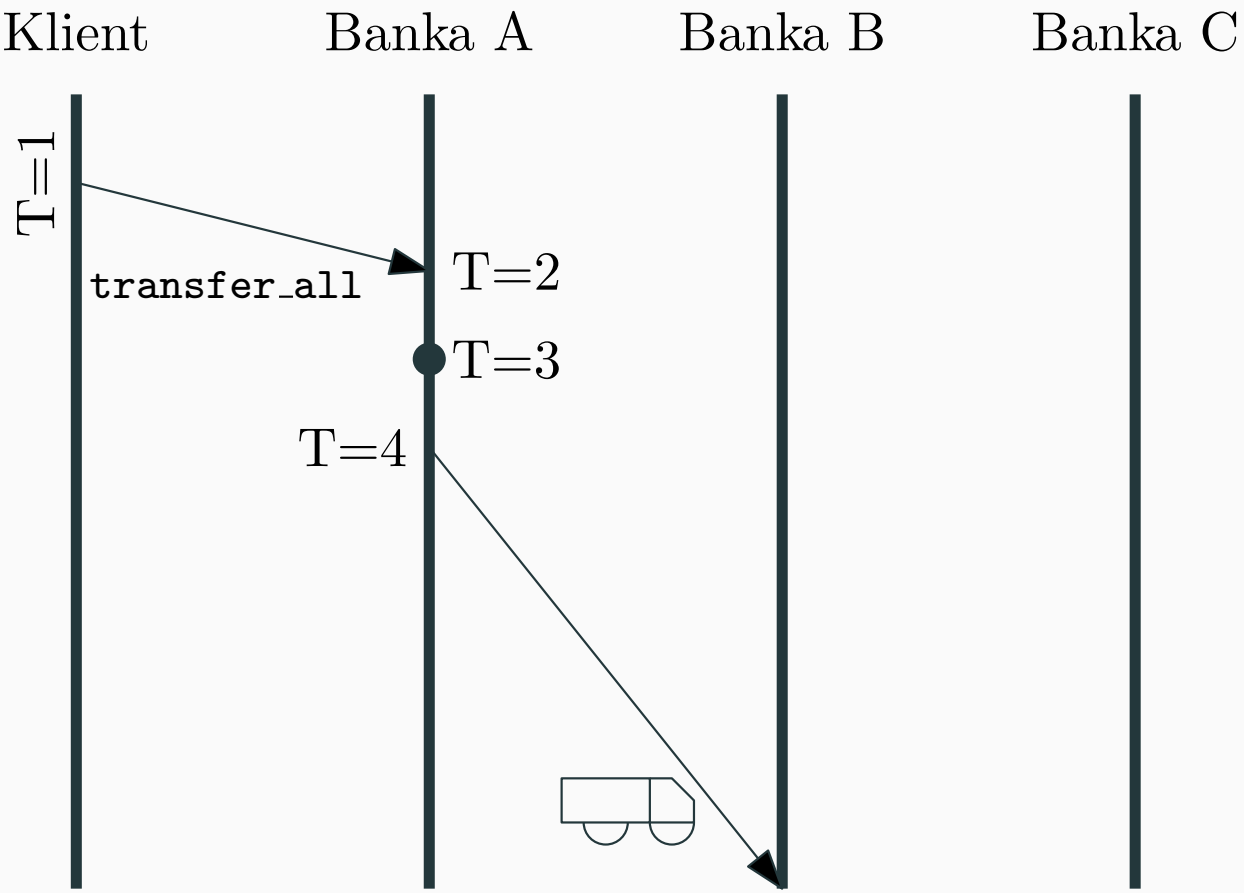
Jak to provést v distribuovaném systému?



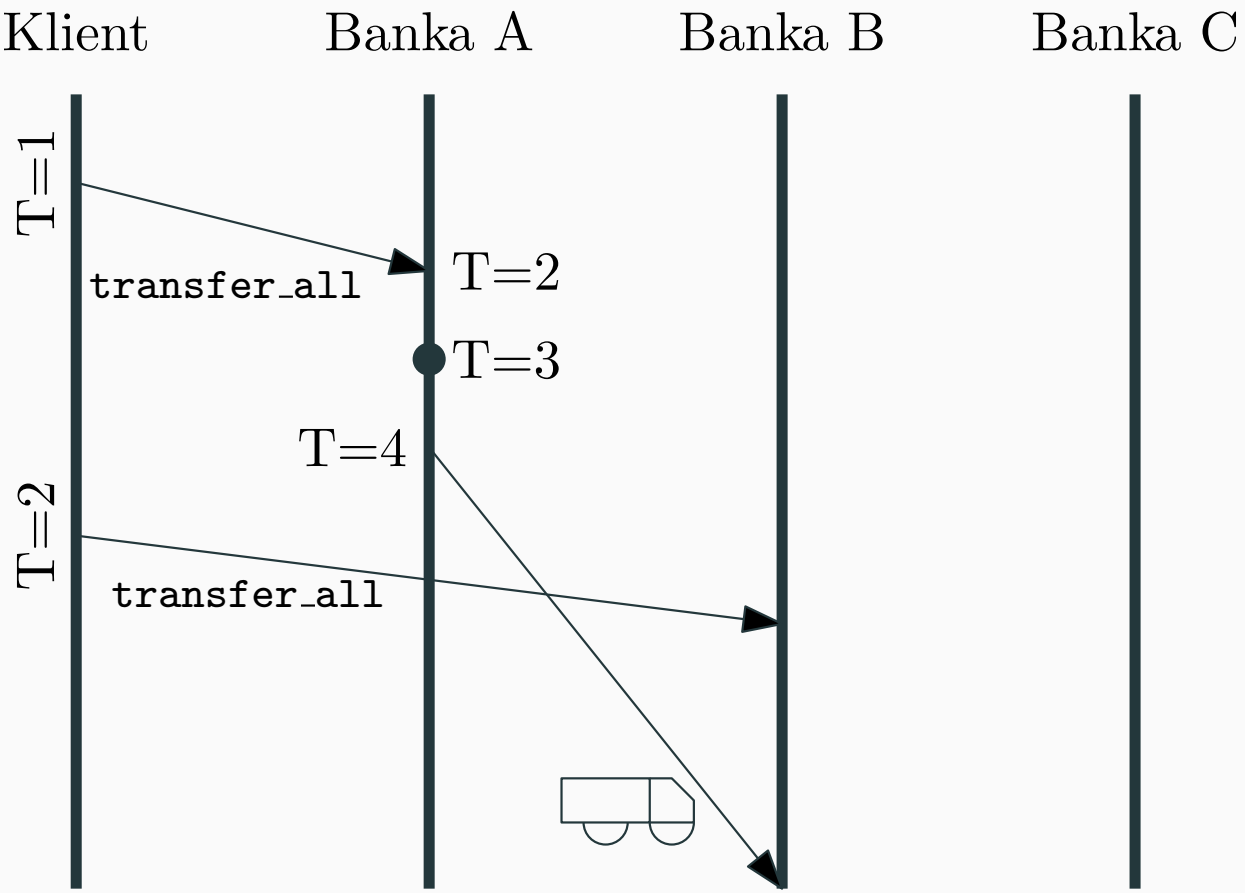
Jak to provést v distribuovaném systému?



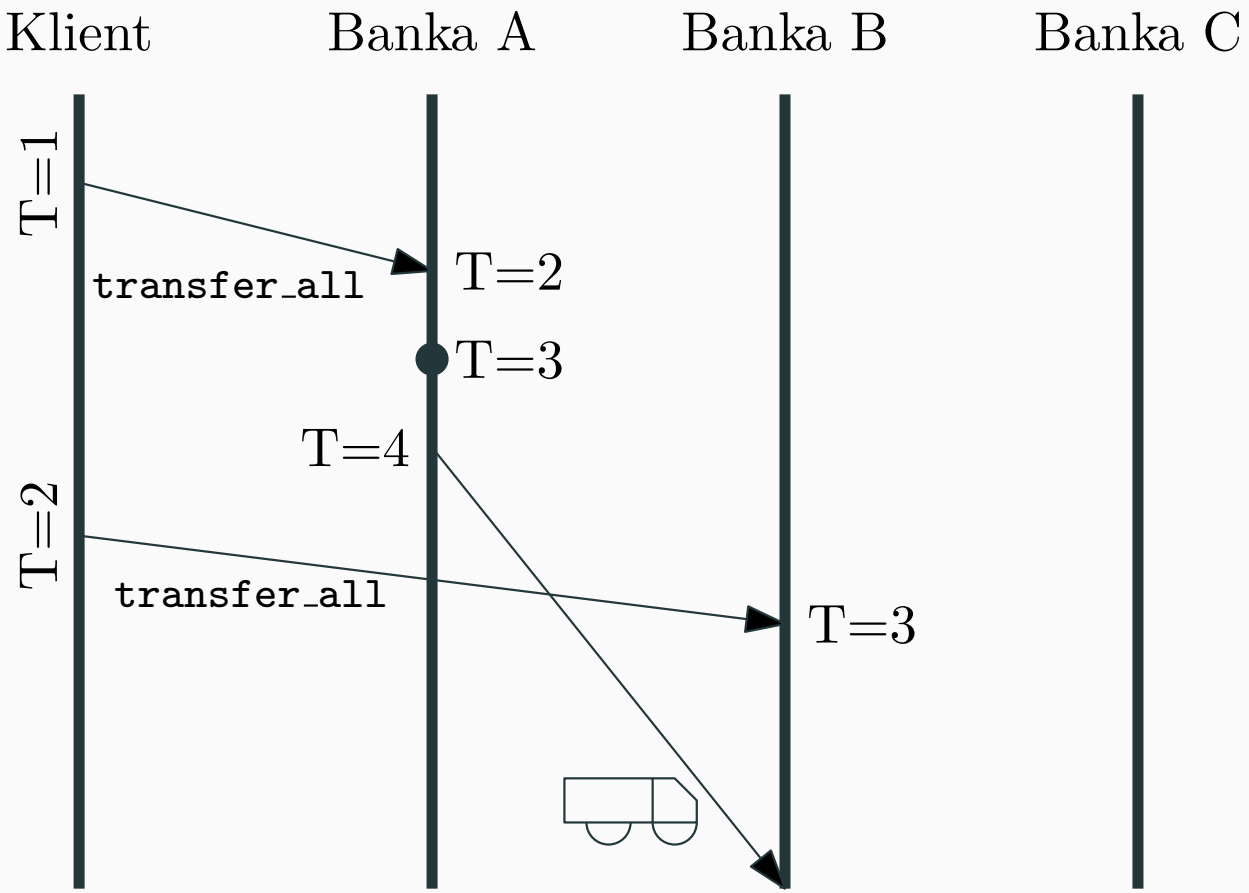
Jak to provést v distribuovaném systému?



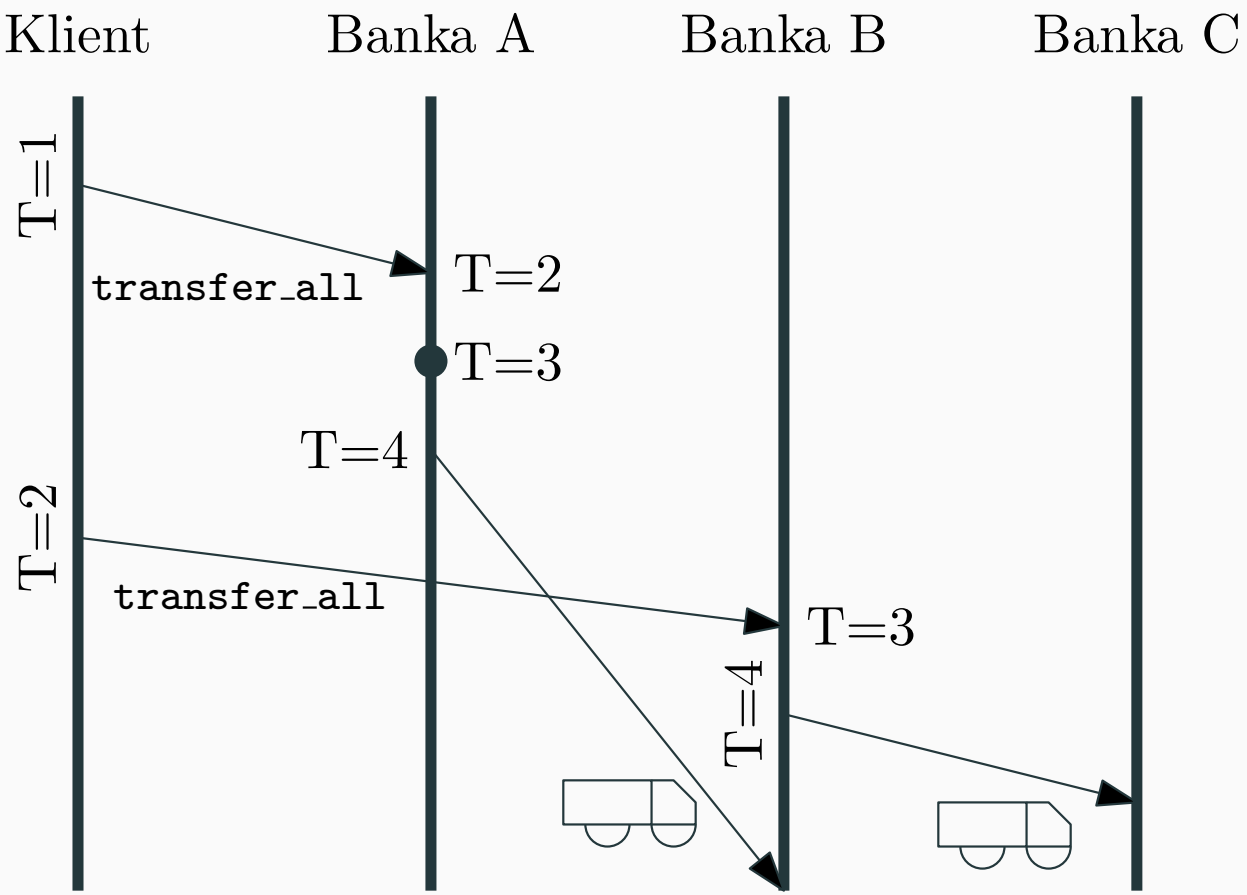
Jak to provést v distribuovaném systému?



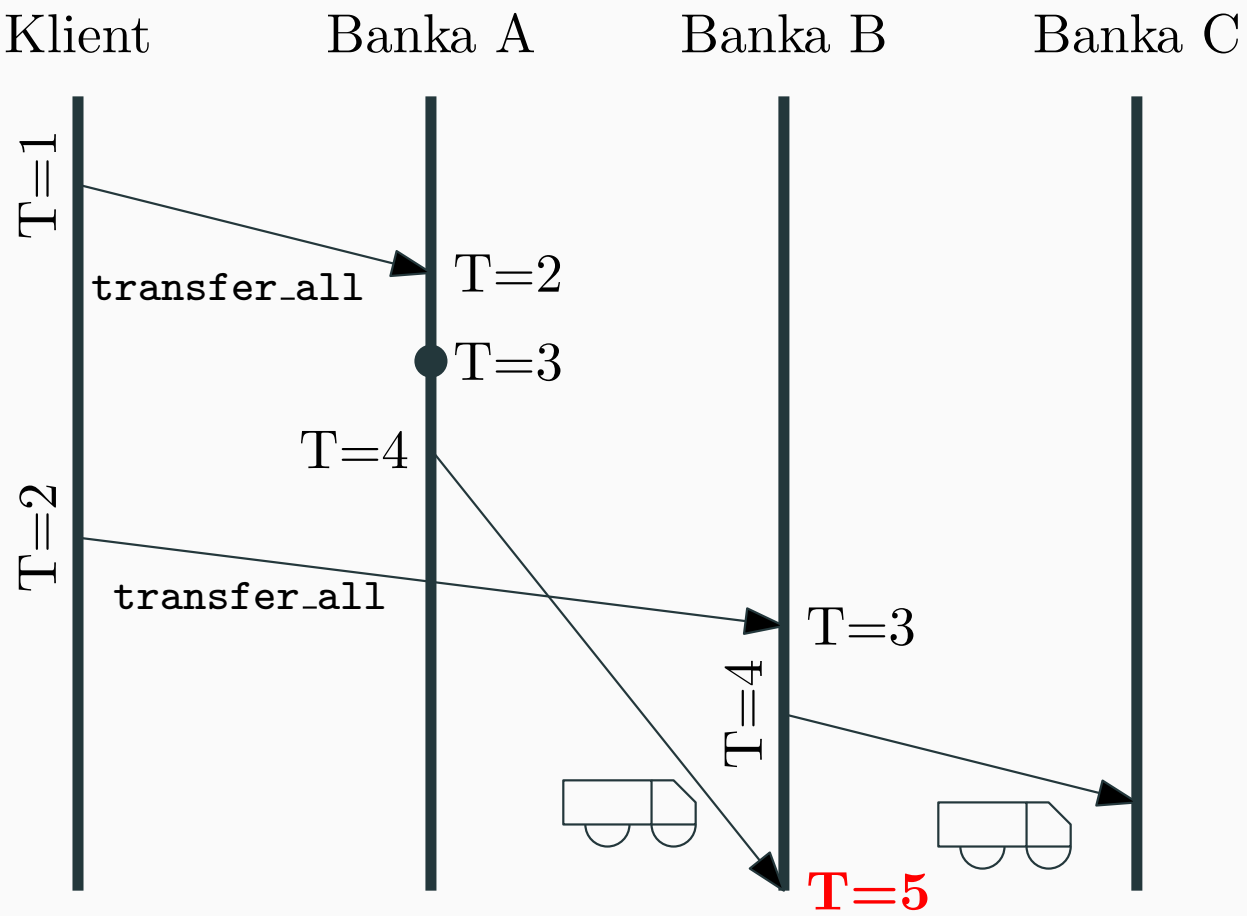
Jak to provést v distribuovaném systému?



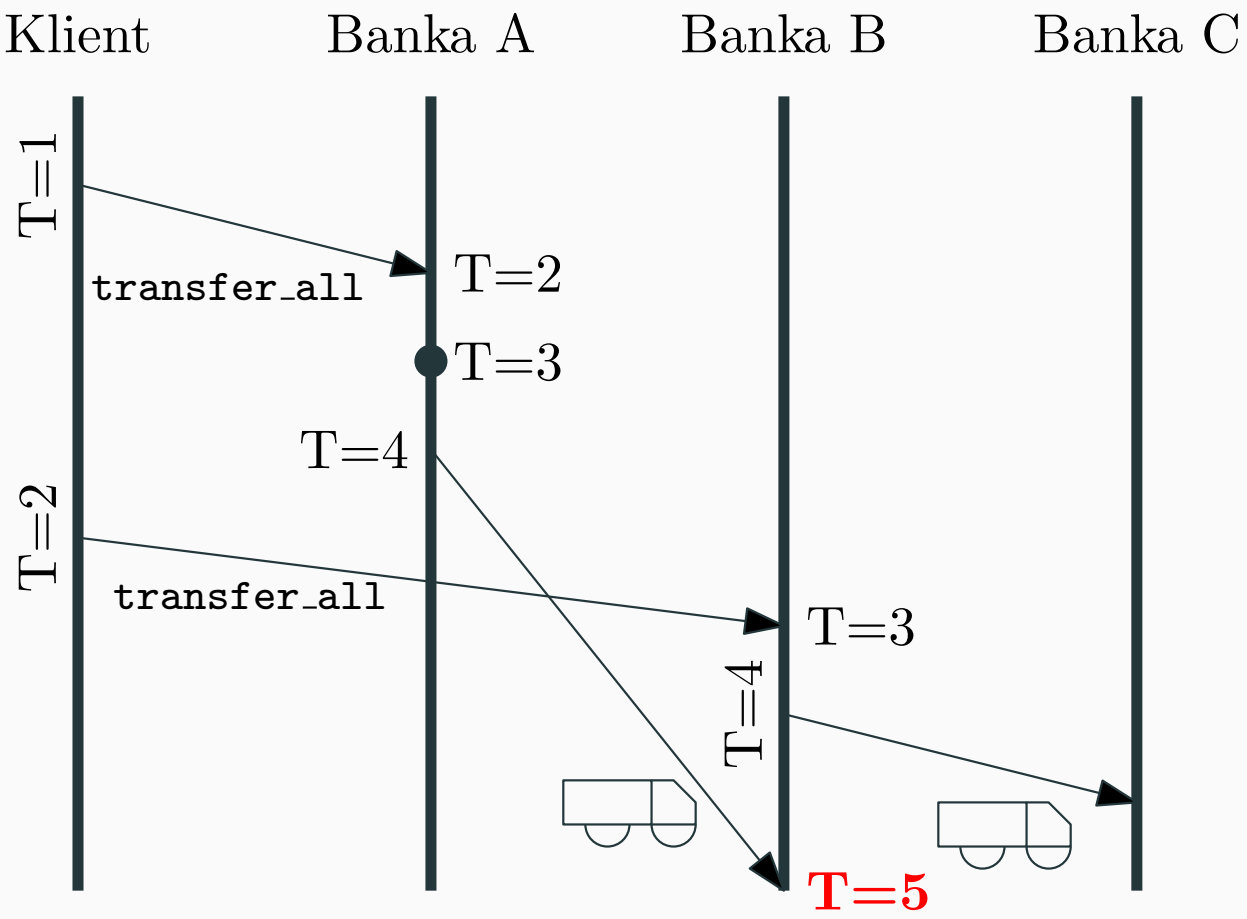
Jak to provést v distribuovaném systému?



Jak to provést v distribuovaném systému?



Jak to provést v distribuovaném systému?



Skalární hodiny agregují všechny události do jediného čísla :-(

Vektorové hodiny

Vektorové hodiny

1. Místo jednoho čísla si držíme vektor časů jednotlivých agentů
   ```
   int[] vectorTime = new int[NUM_AGENTS]
   ```

Vektorové hodiny

1. Místo jednoho čísla si držíme vektor časů jednotlivých agentů
   `int[] vectorTime = new int[NUM_AGENTS]`
2. Před každou významnou událostí (obzvlášť posláním zprávy!) si proces *i* lokální čas posune… **Ale jen svoji komponentu!**
   `++vectorTime[i]`

Vektorové hodiny

1. Místo jednoho čísla si držíme vektor časů jednotlivých agentů
   `int[] vectorTime = new int[NUM_AGENTS]`
2. Před každou významnou událostí (obzvlášť posláním zprávy!) si proces *i* lokální čas posune… **Ale jen svoji komponentu!**
   `++vectorTime[i]`
3. Každé zprávě přiřadíme časovou značku `msg.`T $=$ `vectorTime`

Vektorové hodiny

1. Místo jednoho čísla si držíme vektor časů jednotlivých agentů
   `int[] vectorTime = new int[NUM_AGENTS]`
2. Před každou významnou událostí (obzvlášť posláním zprávy!) si proces i lokální čas posune... **Ale jen svoji komponentu!**
   `++vectorTime[i]`
3. Každé zprávě přiřadíme časovou značku $\texttt{msg.}T = \texttt{vectorTime}$
4. Po přijetí zprávy `msg` procesem i si proces i aktualizuje svůj `logicalTime`:

$$\texttt{vectorTime[j]} = \begin{cases} 1 + \max\{\texttt{vectorTime[j]}, \texttt{msg.}T\texttt{[j]}\} & \text{if } i = j \\ \max\{\texttt{vectorTime[j]}, \texttt{msg.}T\texttt{[j]}\} & \text{jinak} \end{cases}$$

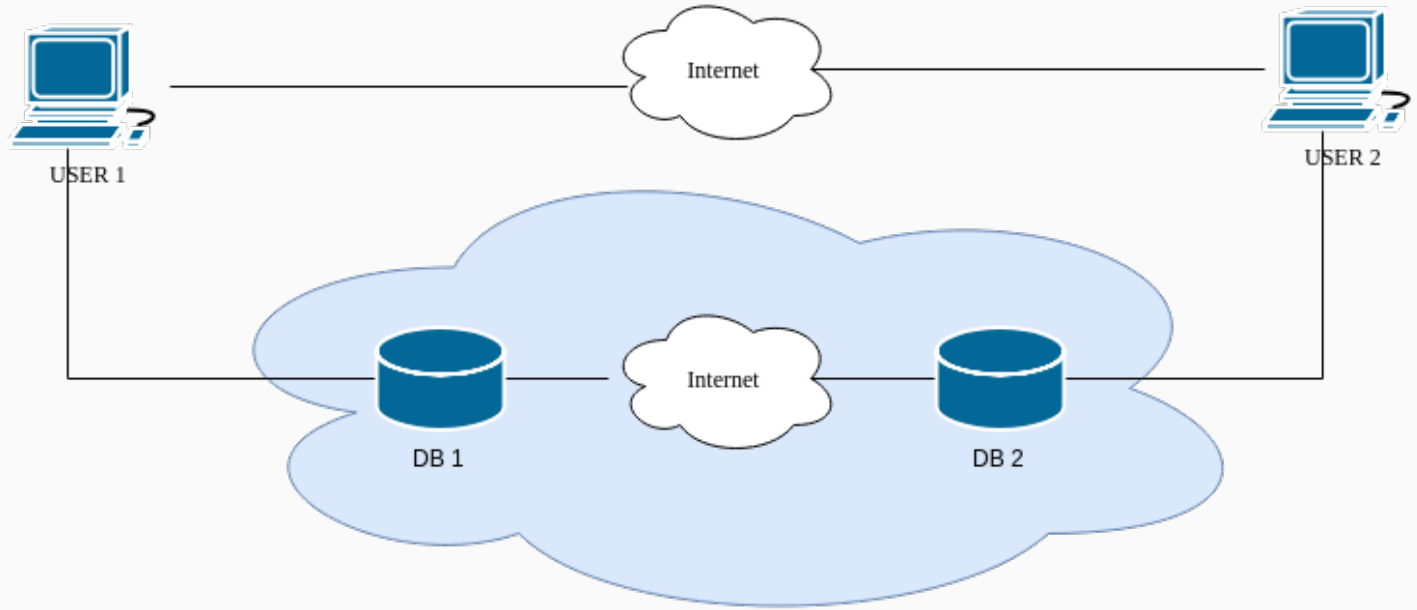
Vektorové hodiny

1. Místo jednoho čísla si držíme vektor časů jednotlivých agentů
   `int[] vectorTime = new int[NUM_AGENTS]`
2. Před každou významnou událostí (obzvlášť posláním zprávy!) si proces i lokální čas posune... **Ale jen svoji komponentu!**
   `++vectorTime[i]`
3. Každé zprávě přiřadíme časovou značku $\texttt{msg.}T = \texttt{vectorTime}$
4. Po přijetí zprávy `msg` procesem i si proces i aktualizuje svůj `logicalTime`:

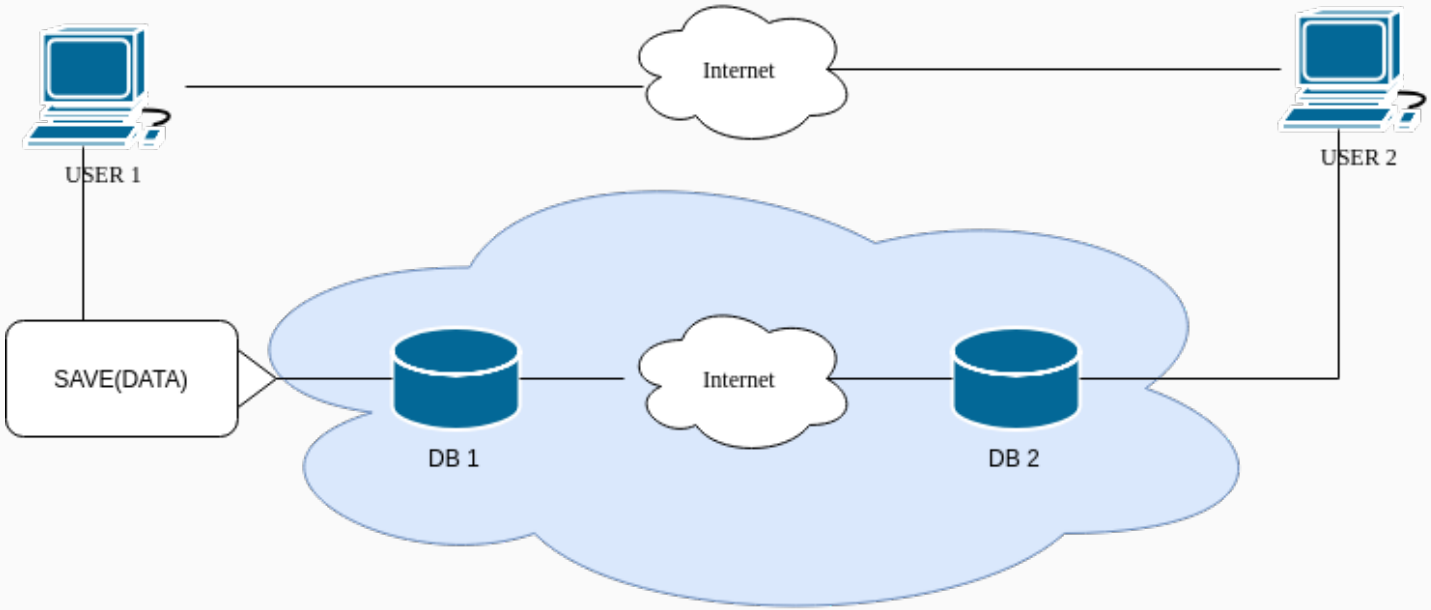
$$\texttt{vectorTime[j]} = \begin{cases} 1 + \max\{\texttt{vectorTime[j]}, \texttt{msg.}T\texttt{[j]}\} & \text{if } i = j \\ \max\{\texttt{vectorTime[j]}, \texttt{msg.}T\texttt{[j]}\} & \text{jinak} \end{cases}$$

⚠ **Vždy posunujeme jen svoji složku časového vektoru!**

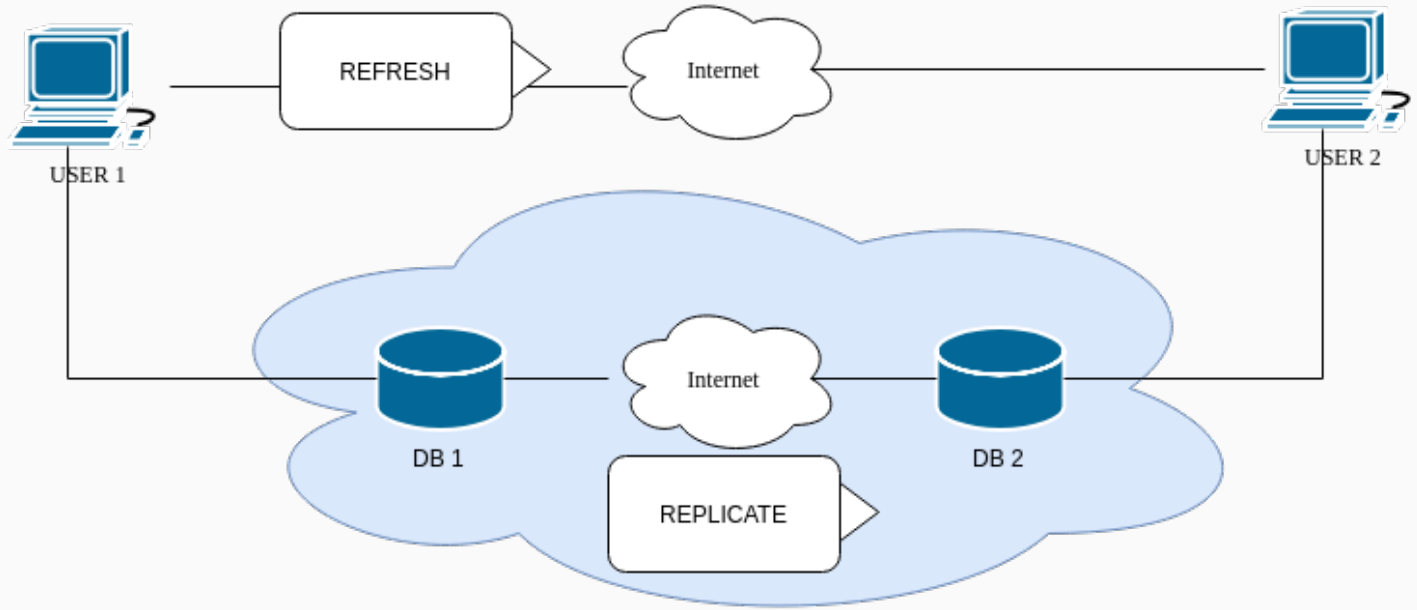
PDV Cloud - připomenutí



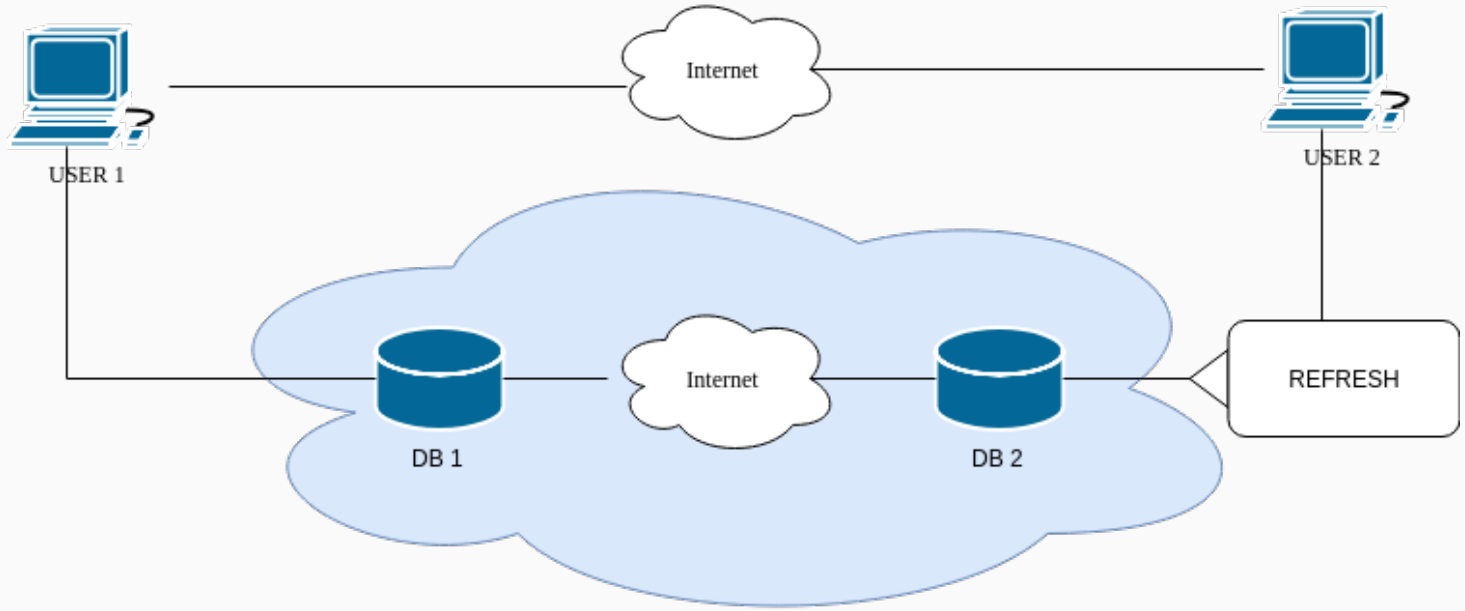
PDV Cloud - připomenutí



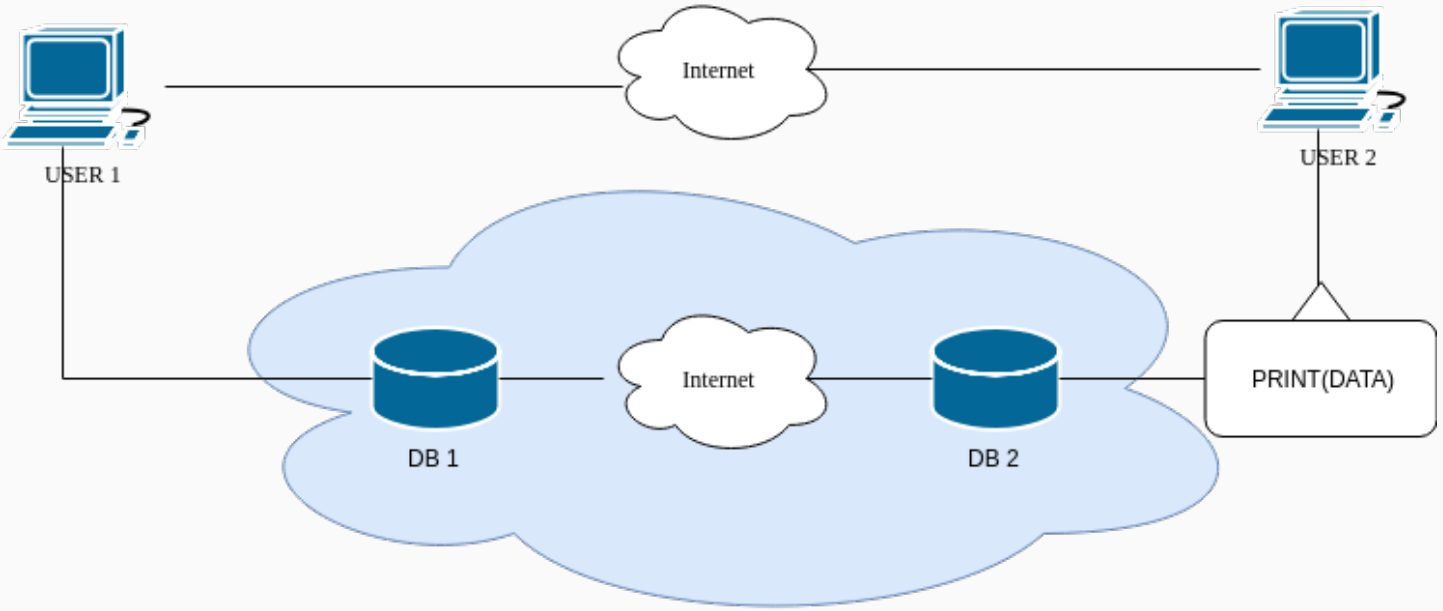
PDV Cloud - připomenutí



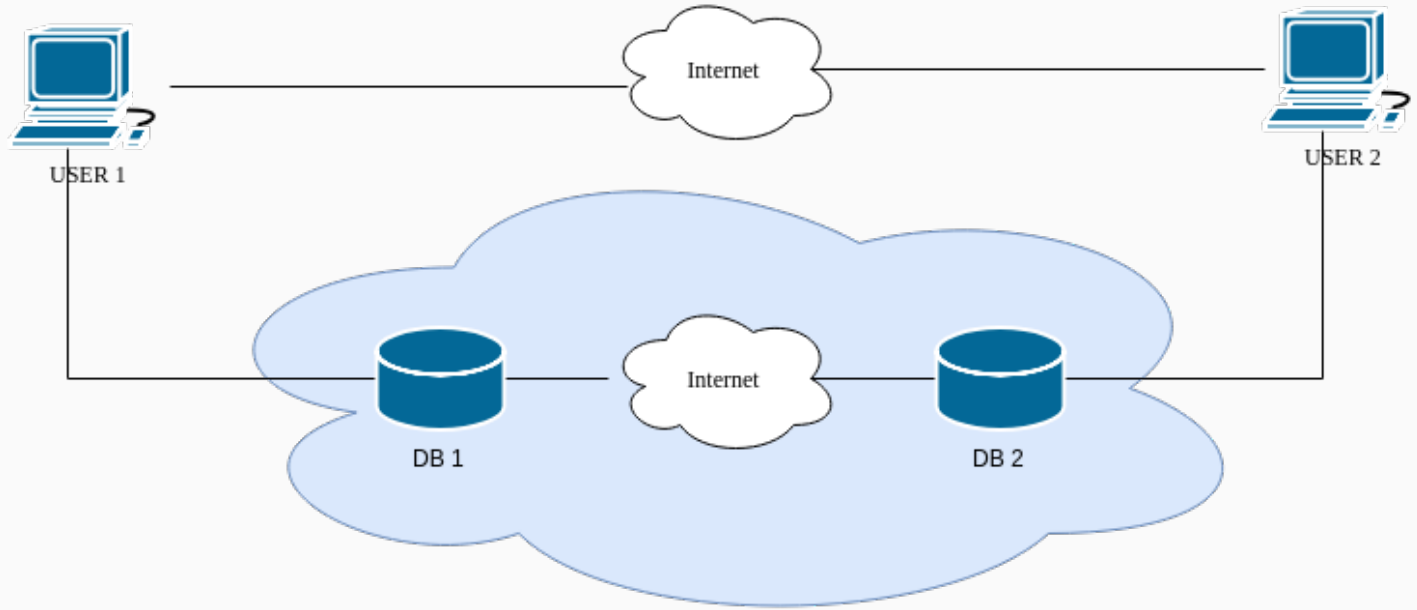
PDV Cloud - připomenutí



PDV Cloud - připomenutí



PDV Cloud - připomenutí



...Doimplementujte metodu `isCausalityForProcessViolated`

Pak zkuste spustit scénář `ScalarDSConfigBombarding` ...

Doprogramujte vektorové logické hodiny

Doimplementujte logiku vektorových logických hodin ve třídě `VectorClock.java`. Následně spusťte scénář `VectorClockRun.java`.

Doprogramujte vektorové logické hodiny

Doimplementujte logiku vektorových logických hodin ve třídě `VectorClock.java`. Následně spusťte scénář `VectorClockRun.java`.

⚠ Jak využít vektorové logické hodiny k detekci souběžných událostí?.

Jak protokol upravit, aby nedocházelo k porušení kauzality?

Jak protokol upravit, aby nedocházelo k porušení kauzality?

Možností je mnoho, například:

Jak protokol upravit, aby nedocházelo k porušení kauzality?

Možností je mnoho, například:

- Před odesláním REFRESH zprávy si počkat na potvrzení od databáze
  (Odeslání REFRESH zprávy je kauzálním následkem úspěšné replikace)

Jak protokol upravit, aby nedocházelo k porušení kauzality?

Možností je mnoho, například:

- Před odesláním REFRESH zprávy si počkat na potvrzení od databáze
  (Odeslání REFRESH zprávy je kauzálním následkem úspěšné replikace)
- Pozdržet vyhodnocení dotazu do doby, než replikace proběhne
  (Druhému uživateli můžeme poslat, že má požadovat data zapsaná nejdříve v daném logickém čase)

Jak protokol upravit, aby nedocházelo k porušení kauzality?

Možností je mnoho, například:

- Před odesláním REFRESH zprávy si počkat na potvrzení od databáze
  (Odeslání REFRESH zprávy je kauzálním následkem úspěšné replikace)
- Pozdržet vyhodnocení dotazu do doby, než replikace proběhne
  (Druhému uživateli můžeme poslat, že má požadovat data zapsaná nejdříve v daném logickém čase)

⚠ **Obecně chceme, aby události e_1, e_2, které mají proběhnout po sobě (tj. například čtení až po replikaci) byly ve vztahu kauzální závislosti.**

Vzájemné vyloučení

S přístupem více vláken k **jednomu zdroji** jsme se již setkali

→ Musíme zaručit konzistenci zdroje

Např. v **OpenMp** pomocí `#pragma omp critical`

S přístupem více vláken k **jednomu zdroji** jsme se již setkali

→ Musíme zaručit konzistenci zdroje

Např. v **OpenMp** pomocí `#pragma omp critical`

Jak to vyřešit v případě DS?

Nejjednodušší možnost: O zdroj se stará samostatný proces

→ Běží na samostatném stroji

→ Může použít vlastní způsoby synchronizace

Nejjednodušší možnost: O zdroj se stará samostatný proces

→ Běží na samostatném stroji

→ Může použít vlastní způsoby synchronizace

V čem je tedy problém?

Nejjednodušší možnost: O zdroj se stará samostatný proces

→ Běží na samostatném stroji

→ Může použít vlastní způsoby synchronizace

V čem je tedy problém?

Některé praktické případy DS toto **neumožňují**

- Požadujeme bezstavovost zdroje
  (souborové NFS servery)
- Zdroj nemá výpočetní jednotku
  (sítě Ethernet a IEEE 802.11, procesy přistupují k jednomu výstupnímu komunikačnímu kanálu)
- … a jiné

Požadavky na vzájemné vyloučení

U procesů máme podobné požadavky jako u vláken

Požadavky na vzájemné vyloučení

U procesů máme podobné požadavky jako u vláken

- **Safety**: v každém okamžiku ke zdroji přistupuje nanejvýš jeden proces

Požadavky na vzájemné vyloučení

U procesů máme podobné požadavky jako u vláken

- **Safety**: v každém okamžiku ke zdroji přistupuje nanejvýš jeden proces
- **Liveness**: každá žádost o přístup ke zdroji je splněna v konečném čase

Požadavky na vzájemné vyloučení

U procesů máme podobné požadavky jako u vláken

- **Safety**: v každém okamžiku ke zdroji přistupuje nanejvýš jeden proces
- **Liveness**: každá žádost o přístup ke zdroji je splněna v konečném čase
- Fairness: procesy získávají přístup k pořadí, v jakém o něj požádali

---

A hodnotíme je podobným způsobem

Požadavky na vzájemné vyloučení

U procesů máme podobné požadavky jako u vláken

- **Safety**: v každém okamžiku ke zdroji přistupuje nanejvýš jeden proces
- **Liveness**: každá žádost o přístup ke zdroji je splněna v konečném čase
- Fairness: procesy získávají přístup k pořadí, v jakém o něj požádali

---

A hodnotíme je podobným způsobem

- Kolik zpráv je nutné si vyměnit, aby došlo k získání a poté uvolnění zdroje?

Požadavky na vzájemné vyloučení

U procesů máme podobné požadavky jako u vláken

- **Safety**: v každém okamžiku ke zdroji přistupuje nanejvýš jeden proces
- **Liveness**: každá žádost o přístup ke zdroji je splněna v konečném čase
- Fairness: procesy získávají přístup k pořadí, v jakém o něj požádali

---

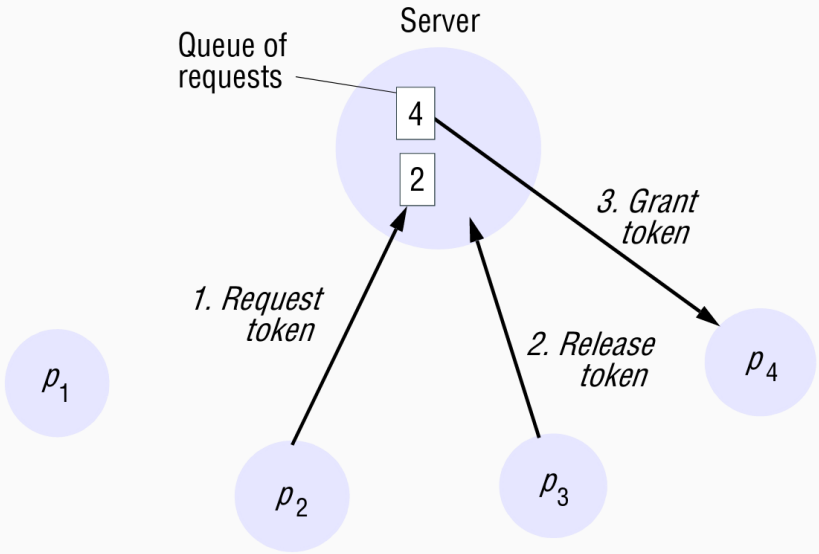
A hodnotíme je podobným způsobem

- Kolik zpráv je nutné si vyměnit, aby došlo k získání a poté uvolnění zdroje?
- Kdy nejdříve po uvolnění může zdroj získat další proces?

Jaké možnosti tedy v DS máme?

Centrální server

Jeden z procesů je určený jako správce požadavků

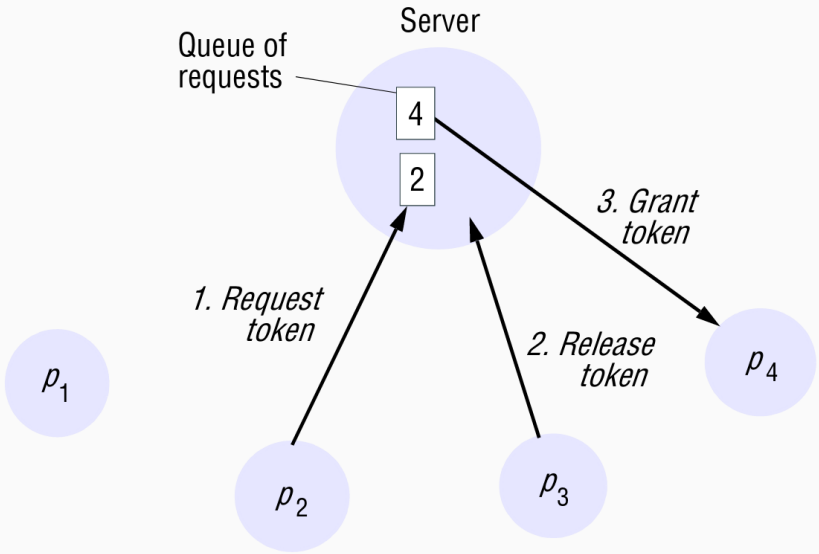


Udržuje si frontu doručených požadavků

Přiznává přístup ke zdroji v pořadí daném frontou

Centrální server

Jeden z procesů je určený jako správce požadavků



Udržuje si frontu doručených požadavků

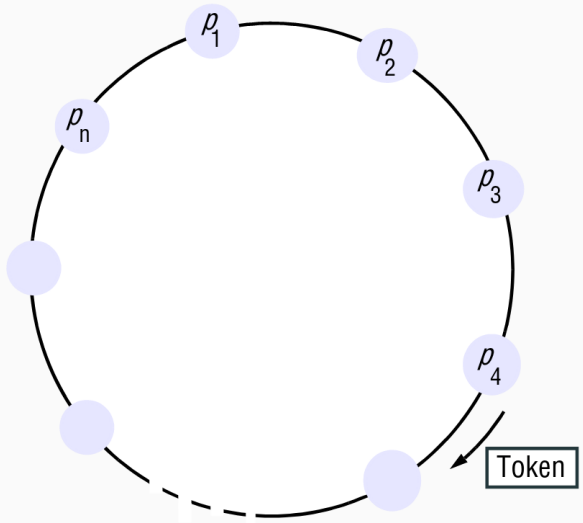
Přiznává přístup ke zdroji v pořadí daném frontou

:(To jsme si moc nepomohli (zavedli jsme single point of failure)

Navíc není splněn požadavek o zachování pořadí (pořadí závisí na latenci komunikace)

Kruhové splňování

Procesy jsou uspořádané v kruhu



Posílají si povolení k přístupu ke zdroji

Jakmile proces zdroj již nepotřebuje, pošle povolení dál

Opět není splněn požadavek o zachování pořadí

Co použít nějakou techniku kterou již známe?

Co použít nějakou techniku kterou již známe?

Serializaci jsme v DS již využívali. . .

Co použít nějakou techniku kterou již známe?

Serializaci jsme v DS již využívali. . .

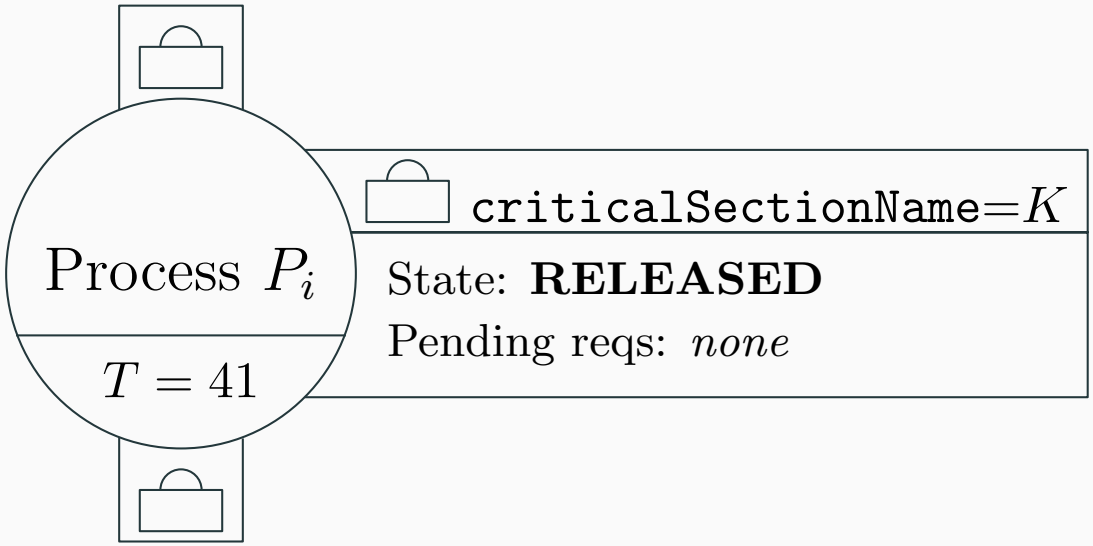
Hodiny!

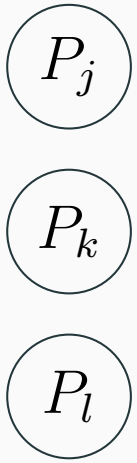
Jak je tedy konkrétně použít?

Ricart-Agrawalovo vyloučení

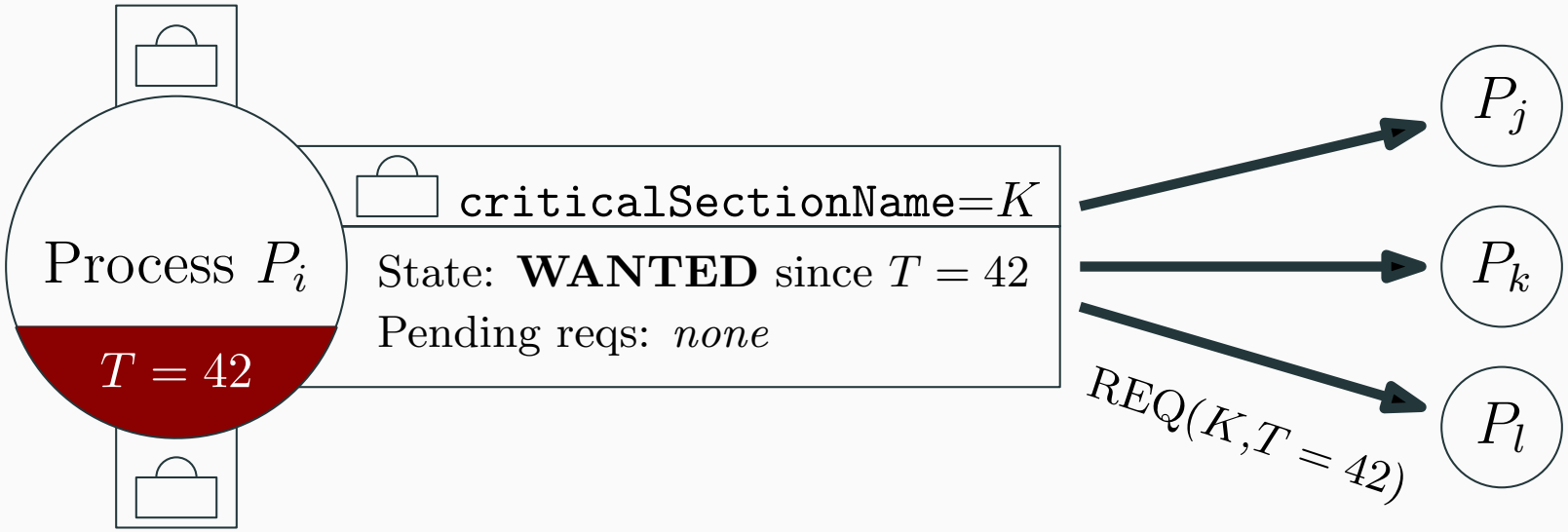
Ricart, Agrawal: An Optimal Algorithm for Mutual Exclusion in Computer Networks, 1981

Žádost o vstup do kritické sekce



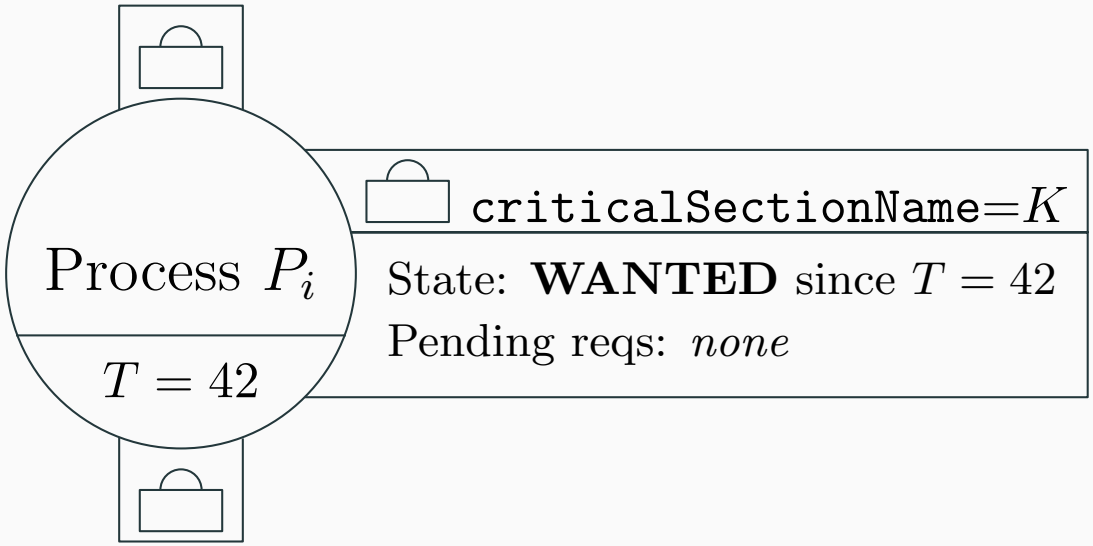


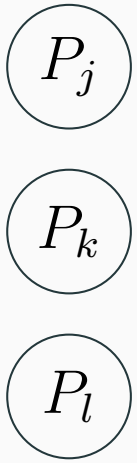
Žádost o vstup do kritické sekce



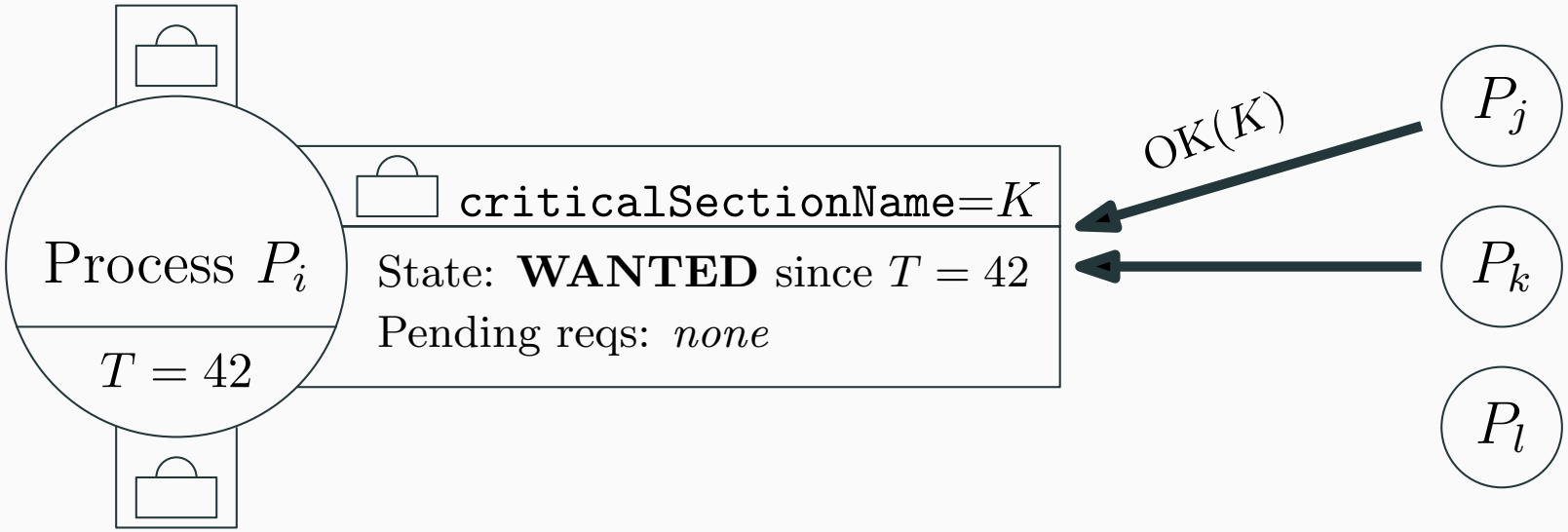
1. Pokud chce proces P_i požádat o vstup do kritické sekce K, zaznamená čas T_i kdy o zdroj žádá a pošle zprávu REQUEST(K) s tímto časem všem procesům, které do K přistupují. Nastaví stav zámku na **WANTED**.

Žádost o vstup do kritické sekce

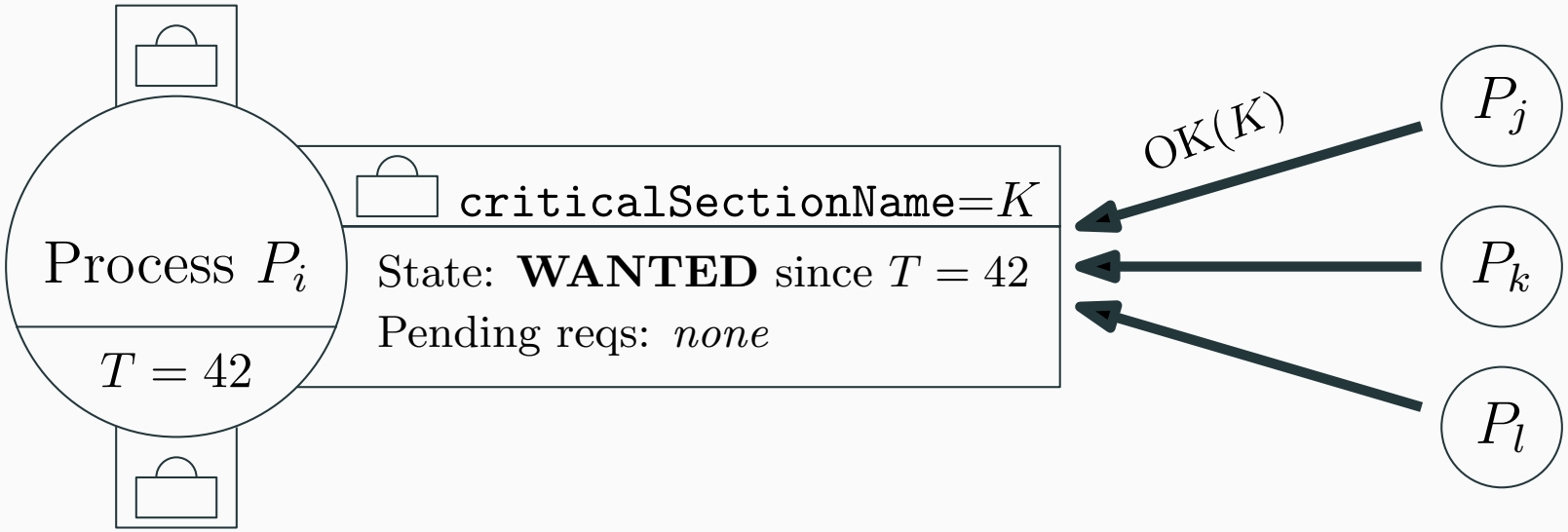




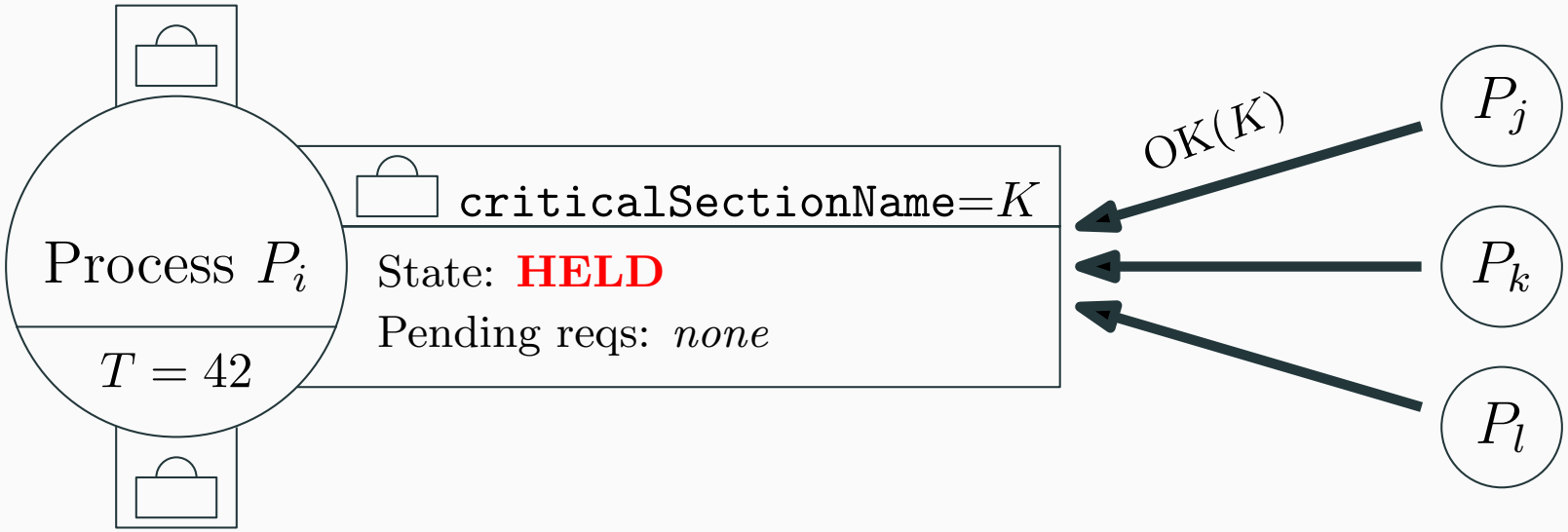
Žádost o vstup do kritické sekce



Žádost o vstup do kritické sekce



Žádost o vstup do kritické sekce

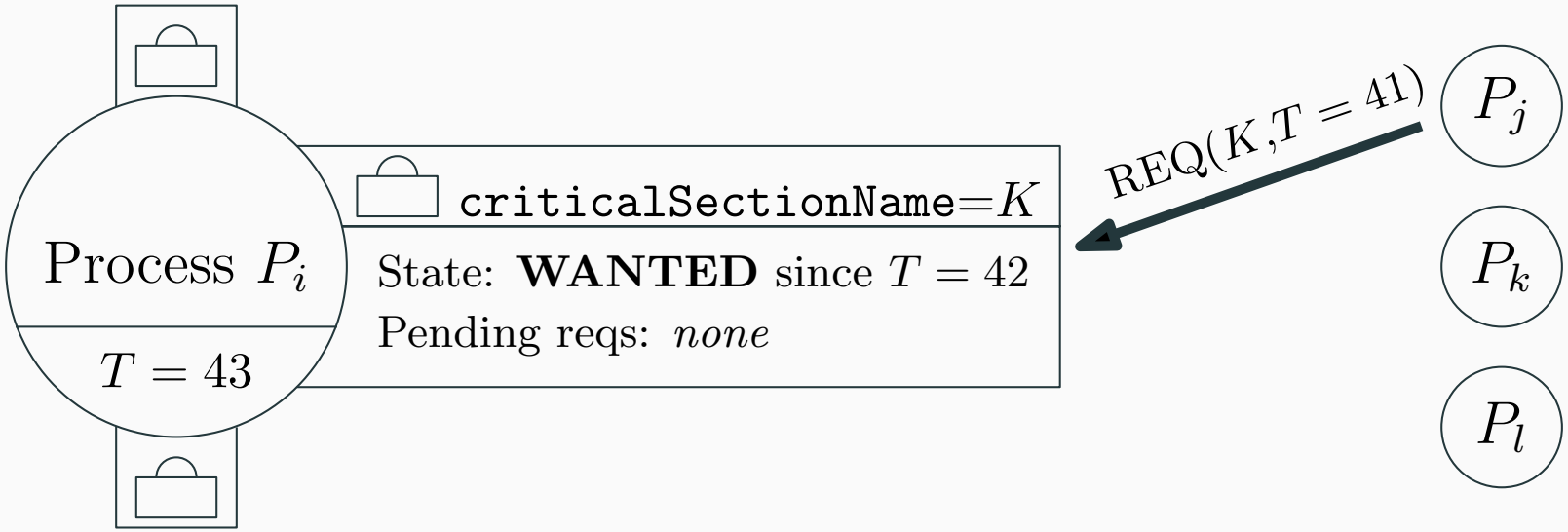


2. Zámek K procesu je ve stavu **WANTED** dokud neobdrží zprávu OK(K) od každého dalšího přistupujícího procesu. Poté se nastaví na **HELD**.

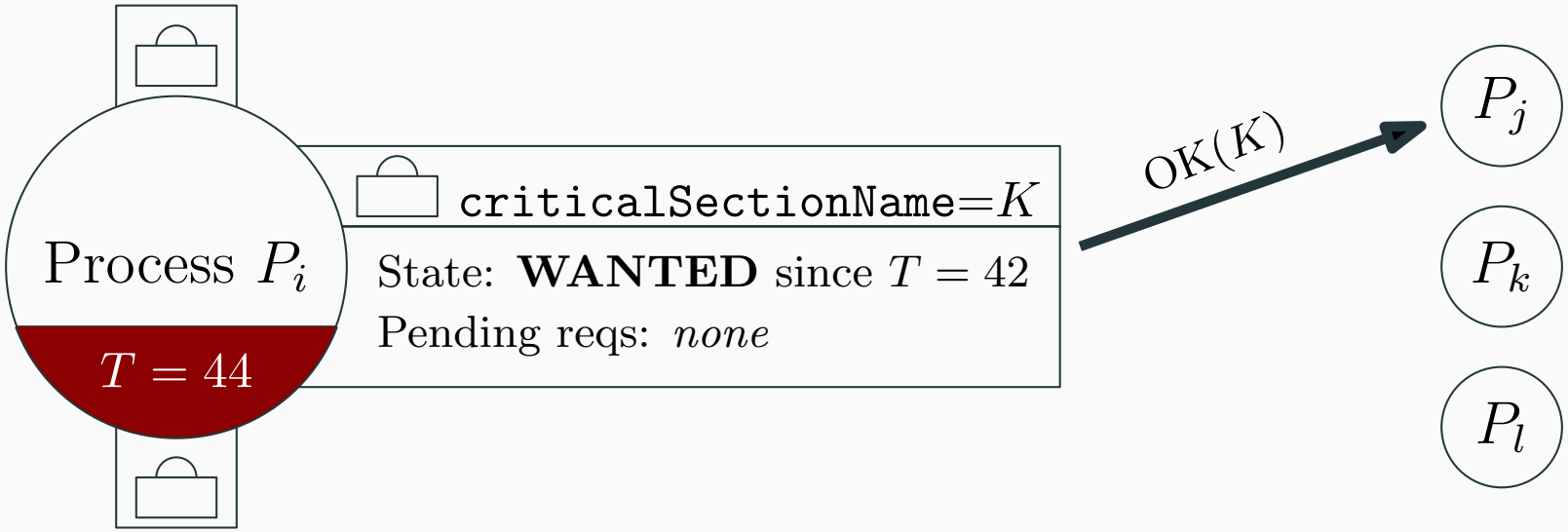
Příchozí požadavek od jiného procesu

Příchozí požadavek od jiného procesu

Příchozí požadavek od jiného procesu



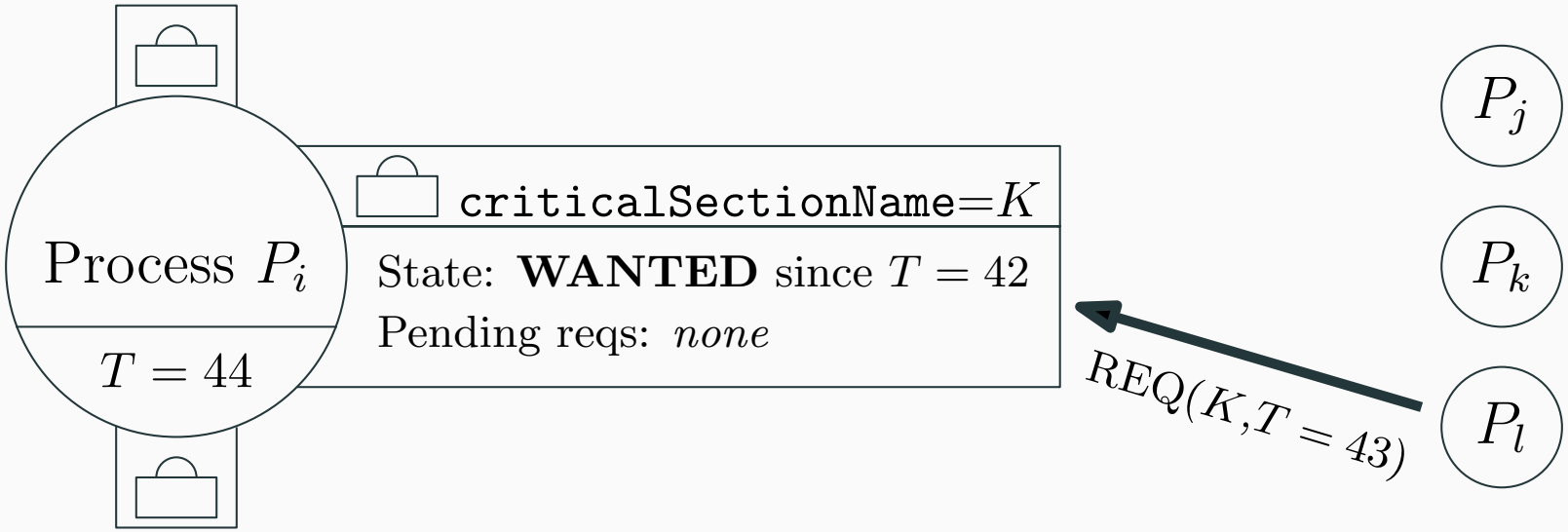
Příchozí požadavek od jiného procesu



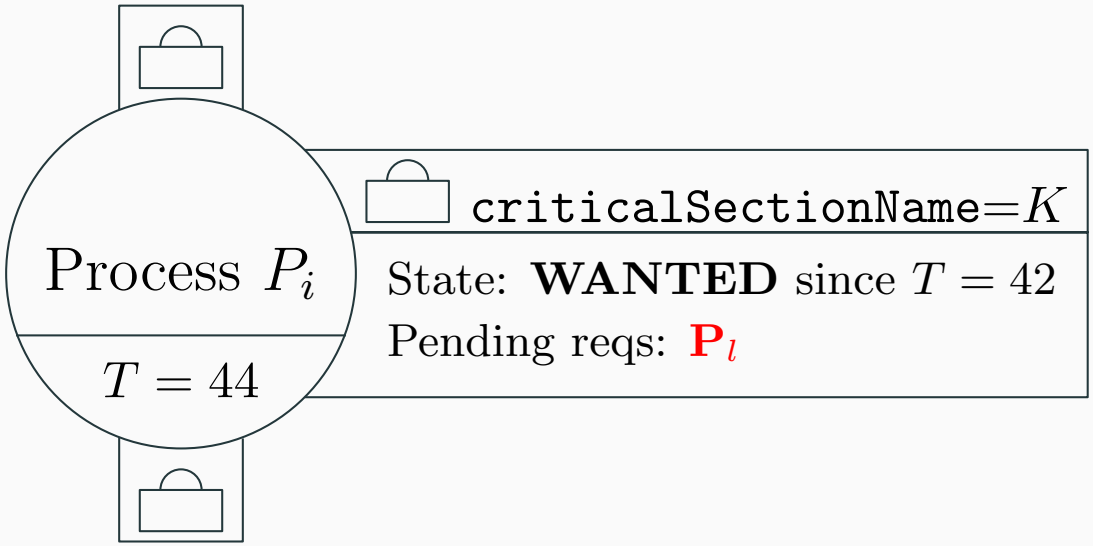
---

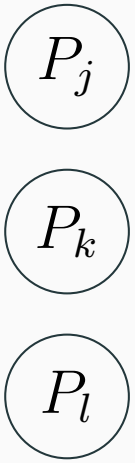
3. Pokud procesu P_i přijde zpráva REQUEST(K) od procesu P_j s časem T_j:
   (i) pokud je zámek K ve stavu **RELEASED**, nebo je ve stavu **WANTED** a o vstup do kritické sekce žádal v čase $T_i > T_j$, pak pošle zprávu OK(K) procesu P_j,

Příchozí požadavek od jiného procesu

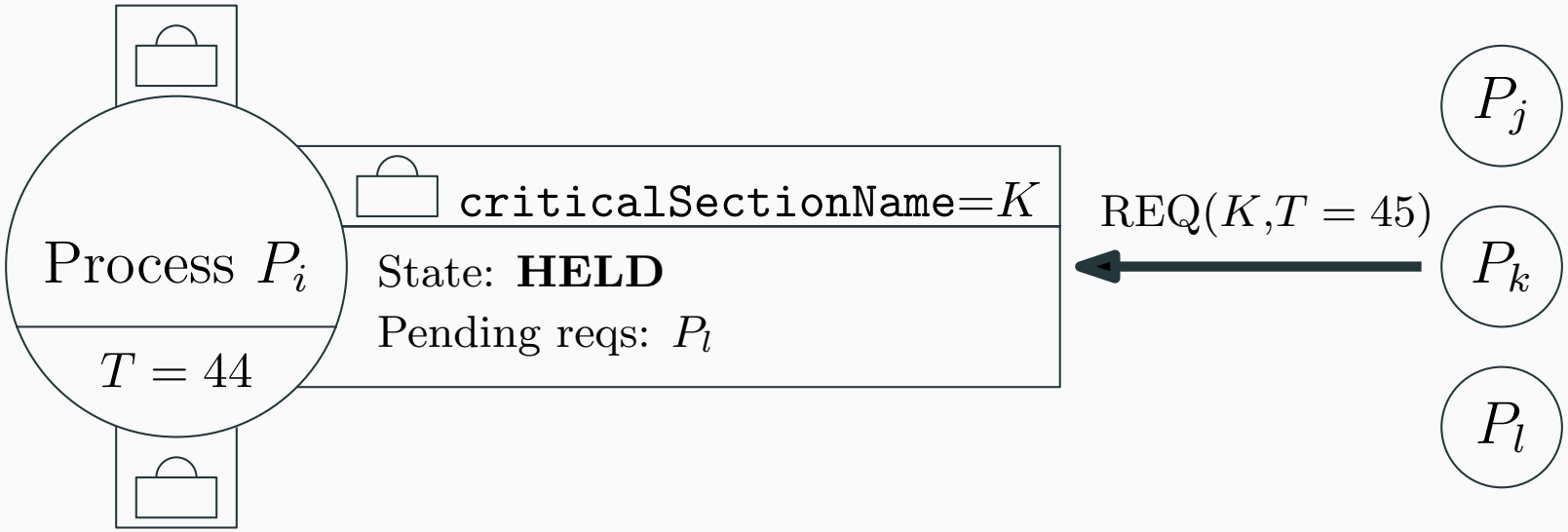


Příchozí požadavek od jiného procesu

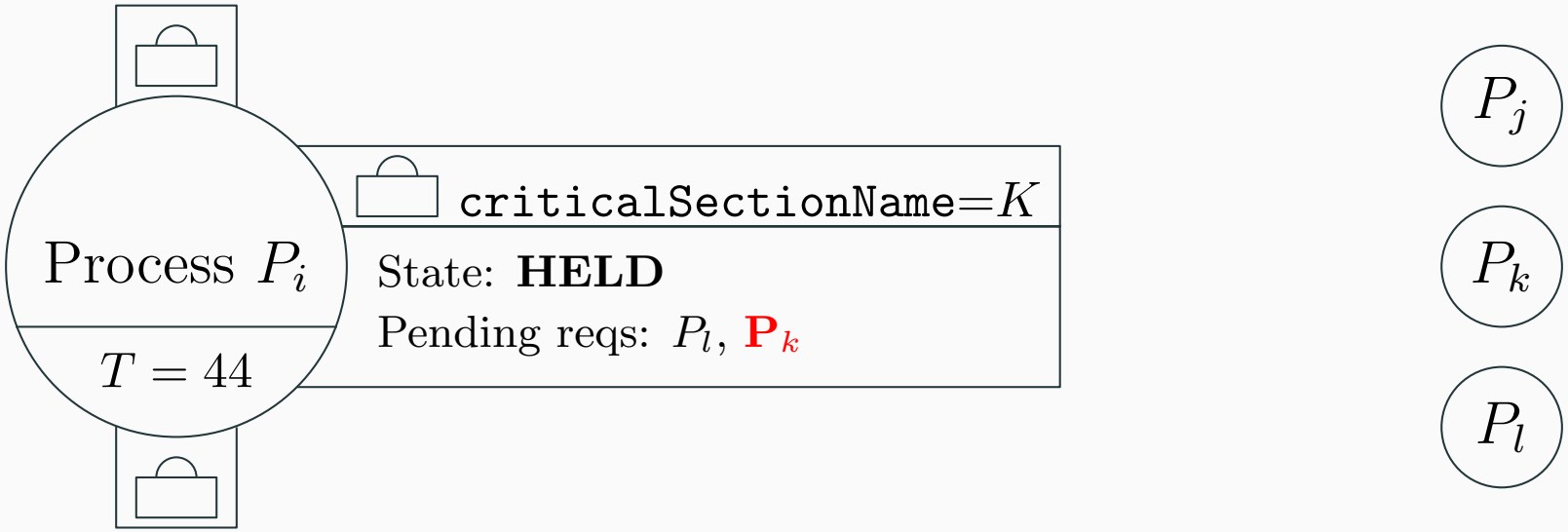




Příchozí požadavek od jiného procesu



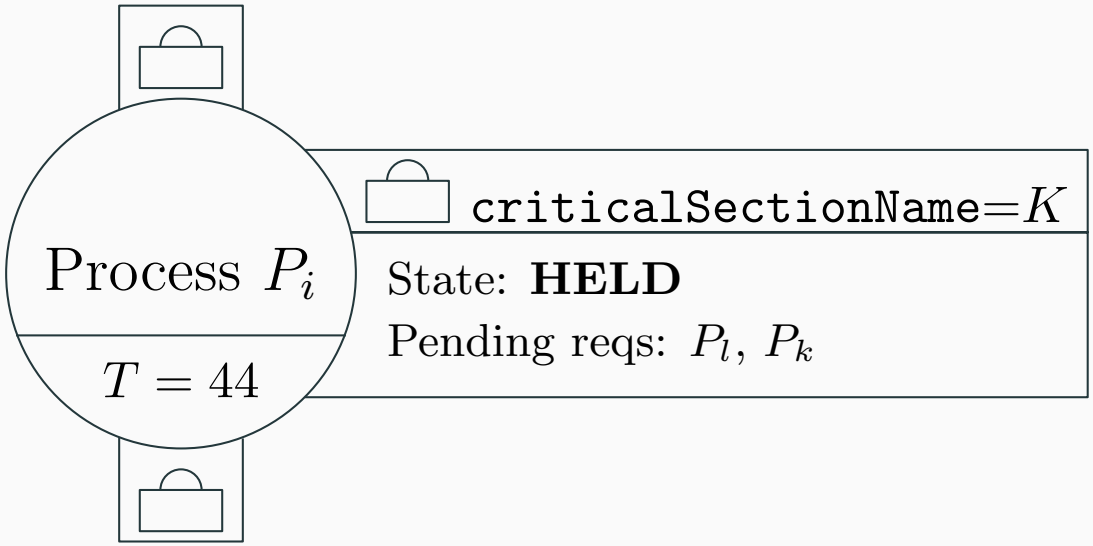
Příchozí požadavek od jiného procesu

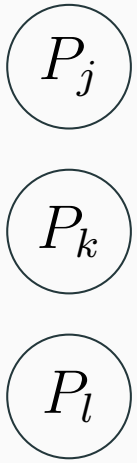


---

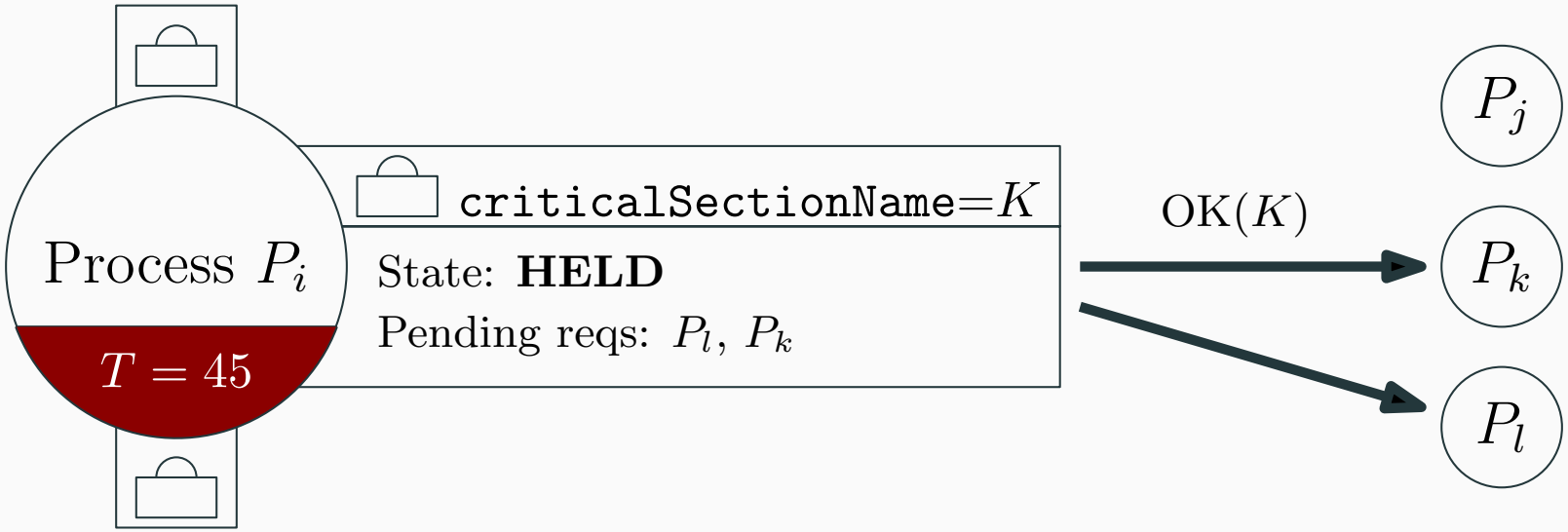
3. Pokud procesu P_i přijde zpráva REQUEST(K) od procesu P_j s časem T_j:
   (ii) jinak požadavek odloží a neodpoví.

Opuštění kritické sekce





Opuštění kritické sekce



Opuštění kritické sekce

---

4. Pokud proces P_i dokončí práci v kritické sekci K, nastaví stav zámku K na **RELEASED**, odpoví na všechny odložené požadavky a frontu požadavků vyprázdní.

Jaké požadavky tento algoritmus splňuje?

1. Safety?
2. Liveness?
3. Fairness?
4. Počet zpráv?

http://goo.gl/a6BEMb

Jaké požadavky tento algoritmus splňuje?

1. Safety? TRUE
2. Liveness?
3. Fairness?
4. Počet zpráv?

Jaké požadavky tento algoritmus splňuje?

1. Safety? TRUE
2. Liveness? TRUE
3. Fairness?
4. Počet zpráv?

Jaké požadavky tento algoritmus splňuje?

1. Safety? TRUE
2. Liveness? TRUE
3. Fairness? TRUE
4. Počet zpráv?

Jaké požadavky tento algoritmus splňuje?

1. Safety? TRUE
2. Liveness? TRUE
3. Fairness? TRUE
4. Počet zpráv? $2(n-1)$

Tolerance k chybám

Dokáží se algoritmy vypořádat se ztrátou dat?

Tolerance k chybám

Dokáží se algoritmy vypořádat se ztrátou dat?

Dokáží se vypořádat s padajícími procesy?

Tolerance k chybám

Dokáží se algoritmy vypořádat se ztrátou dat?

Dokáží se vypořádat s padajícími procesy?

⚠Požadavkem algoritmu je spolehlivá komunikace. Selhávající procesy mohou vést k zamrznutí (bez dodatečných úprav).

Zadání samostatné úlohy

Ricart-Agrawalovo vyloučení

Doprogramujte Ricart-Agrawalův algoritmus

Doimplementujte logiku Ricart-Agrawalova algoritmu ve třídě `exclusion/ExclusionPrimitive.java`. Následně spusťte scénář `bank.Main`.

⚠ **Implementace tohoto zadaní je obsahem 7. domácího úkolu s termínem odevzdání 6. 5. 2022!**

Zpracování musí být **distribuované**, procesy si nesahají vzájemně do paměti!

Díky za pozornost!

Budeme rádi za Vaši zpětnou vazbu! →

https://forms.gle/vwbWazEu14w1Kf487