Vzájemné vyloučení

5. května 2018

B4B36PDV – Paralelní a distribuované výpočty

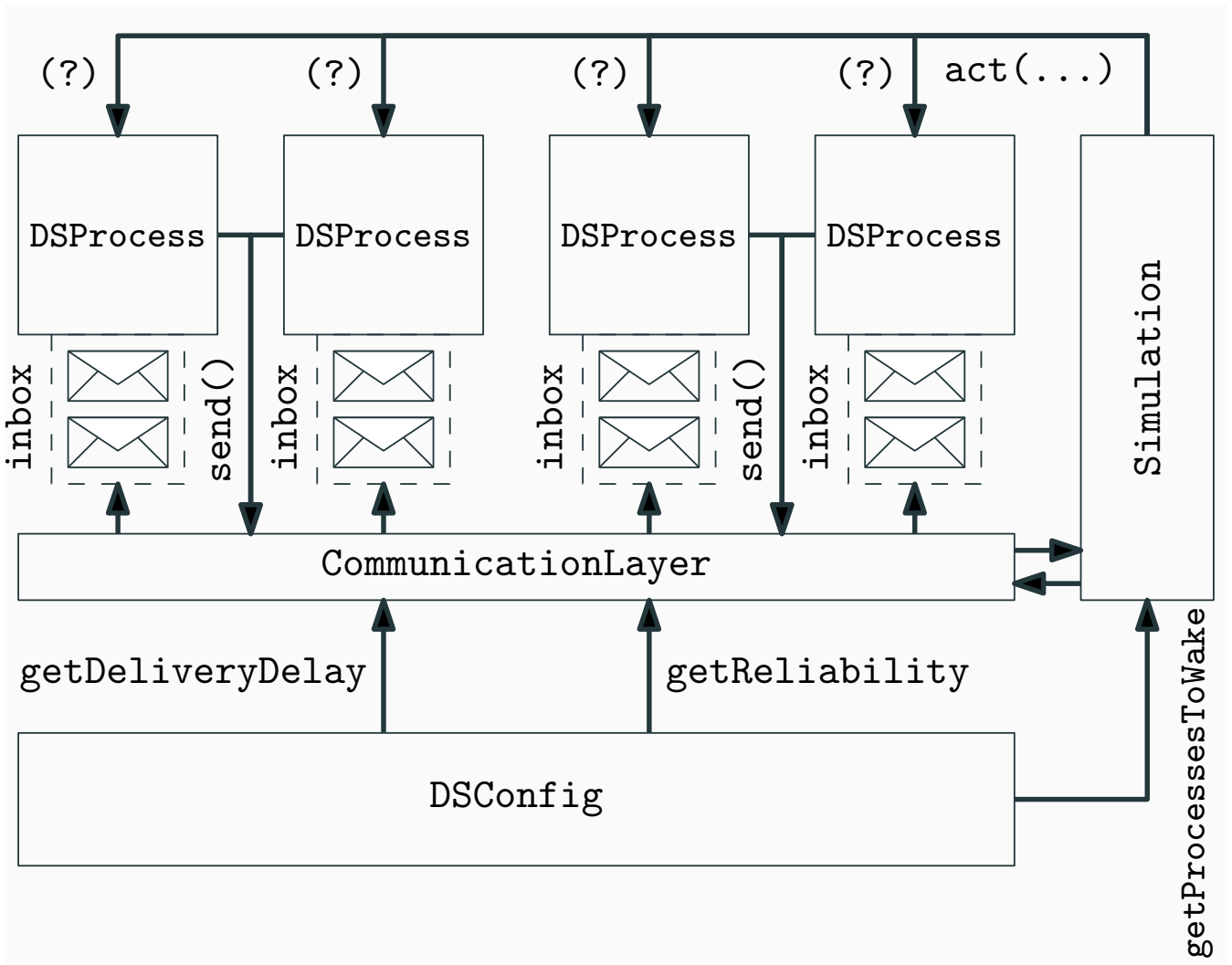
Osnova

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

Opakování z minulého cvičení

http://goo.gl/a6BEMb

DSand framework



Jakou roli hrají v DS logické hodiny?

Zvolte, které z následujících možností platí

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

Jaké vlastnosti mají vektorové hodiny?

Zvolte, které z následujících možností platí

1. jsou paměťově náročnější než logické hodiny
2. dokáží detekovat porušení kauzality vůči konkrétnímu procesu
3. generují částečné uspořádání zpráv
4. určují reálný čas, kdy byla zpráva poslána
5. dokáží detekovat, zda je daná událost kauzálním důsledkem jiné události

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

$\rightarrow$ Běží na samostatném stroji

$\rightarrow$ Může použít vlastní způsoby synchronizace

Nejjednodušší možnost: O zdroj se stará samostatný proces

$\rightarrow$ Běží na samostatném stroji

$\rightarrow$ Může použít vlastní způsoby synchronizace

V čem je tedy problém?

Nejjednodušší možnost: O zdroj se stará samostatný proces

$\rightarrow$ Běží na samostatném stroji

$\rightarrow$ 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 vypoučení

U procesů máme podobné požadavky jako u vláken

Požadavky na vzájemné vypoučení

U procesů máme podobné požadavky jako u vláken

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

Požadavky na vzájemné vypoučení

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 vypoučení

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 vypoučení

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 vypoučení

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

---

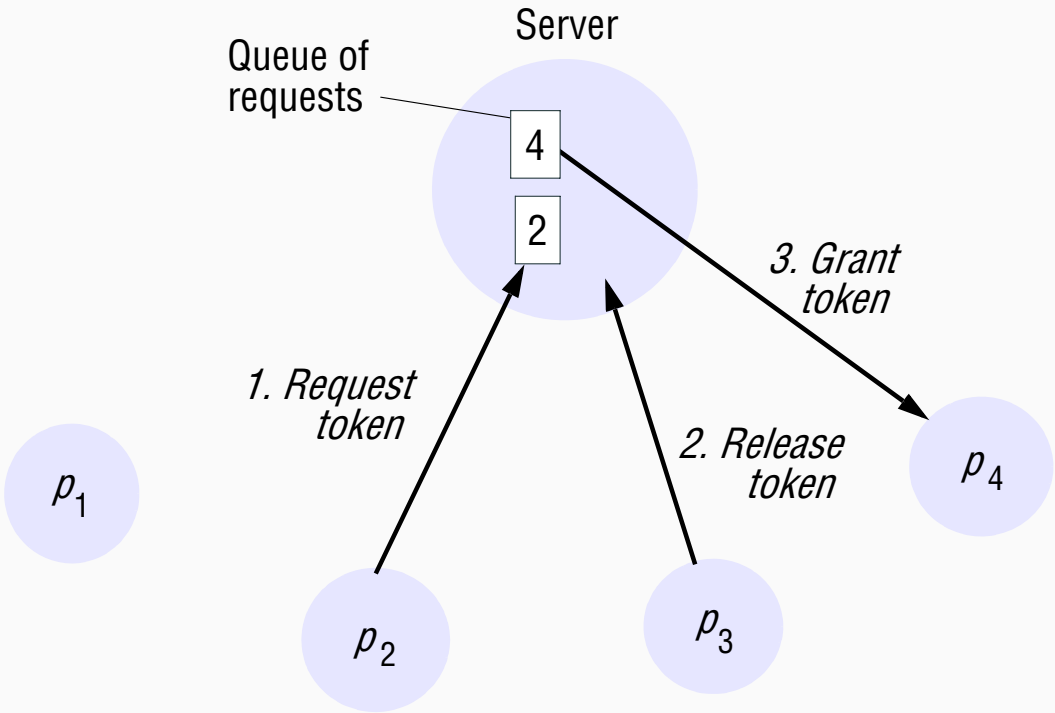
A hodnotíme je podobným způsobem

- 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

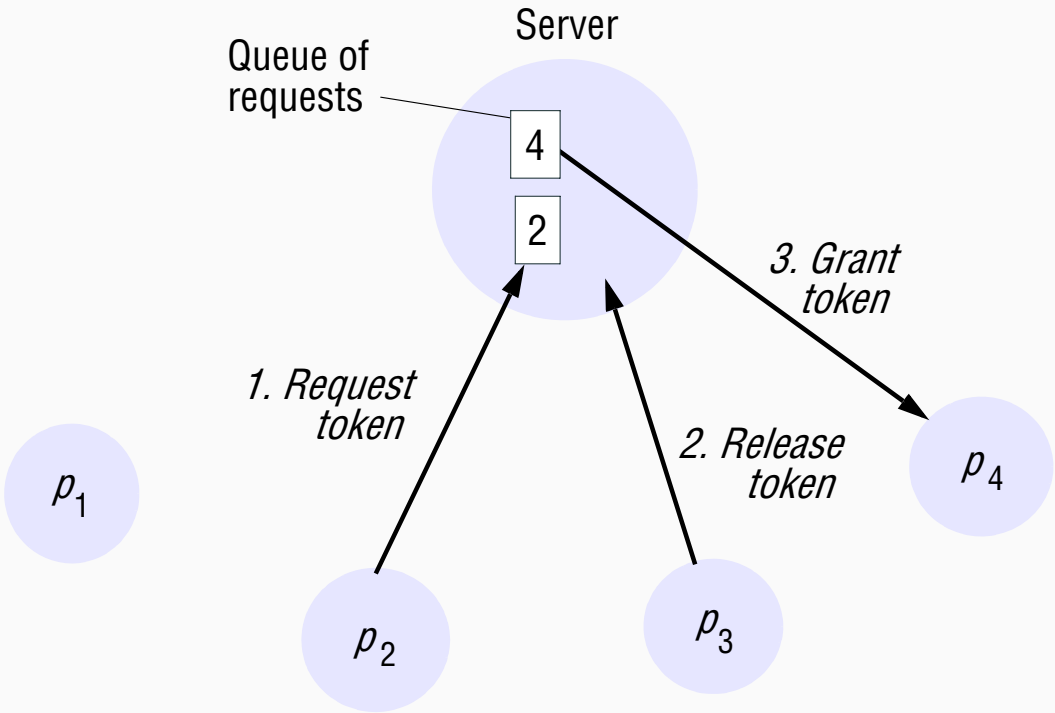


Udržuje si frontu doručených požadavků

Přiznává přístup ke zdroji v pořadí daném frontou

Centrální server

Jeden z procesů je určený jako správce požadavků



Udržuje si frontu doručených požadavků

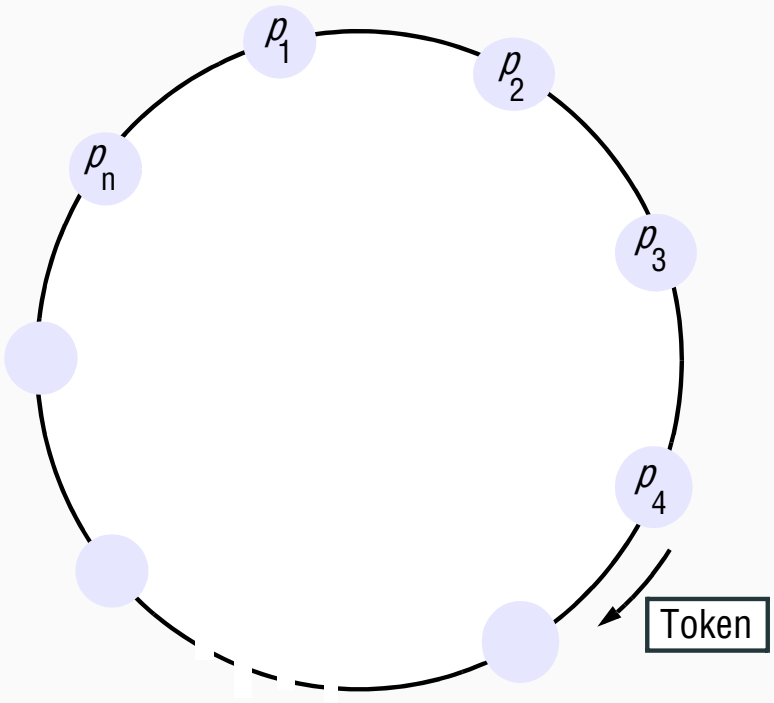
Přiznává přístup ke zdroji v pořadí daném frontou

:(To jsme si moc nepomohli

Navíc není splněn požadavek o zachování pořadí

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. . .

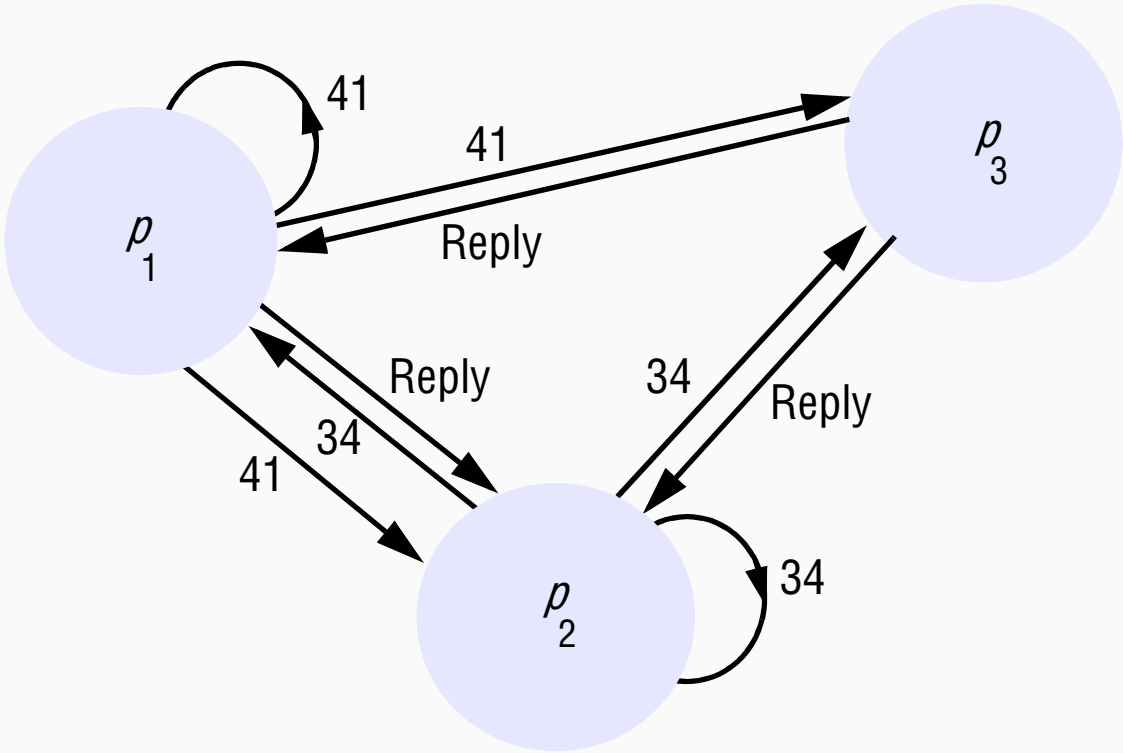
Co použít nějakou techniku kterou již známe?

Serializaci jsme v DS již využívali...

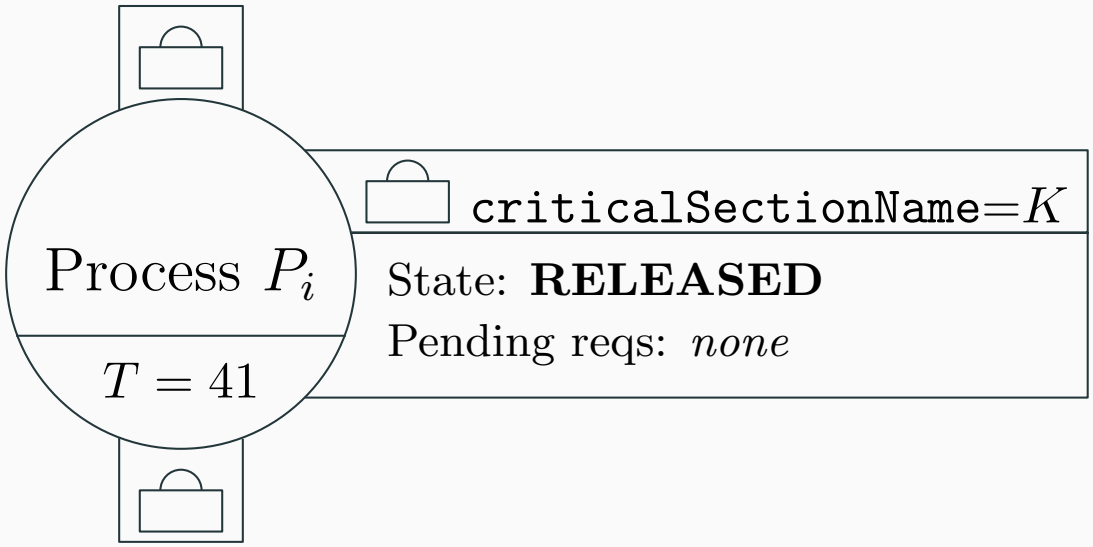
Hodiny!

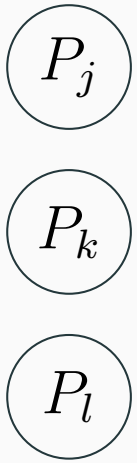
Jak je tedy konkrétně použít?

Ricart-Agrawalovo vyloučení

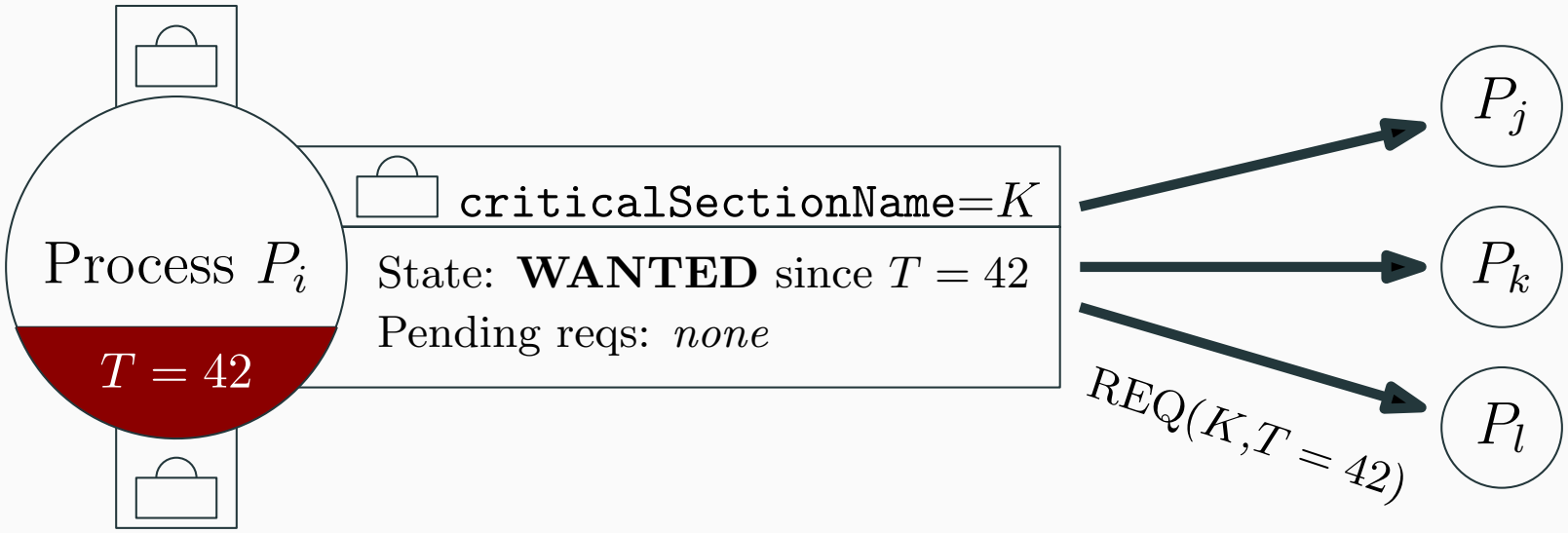


Žádost o vstup do kritické sekce



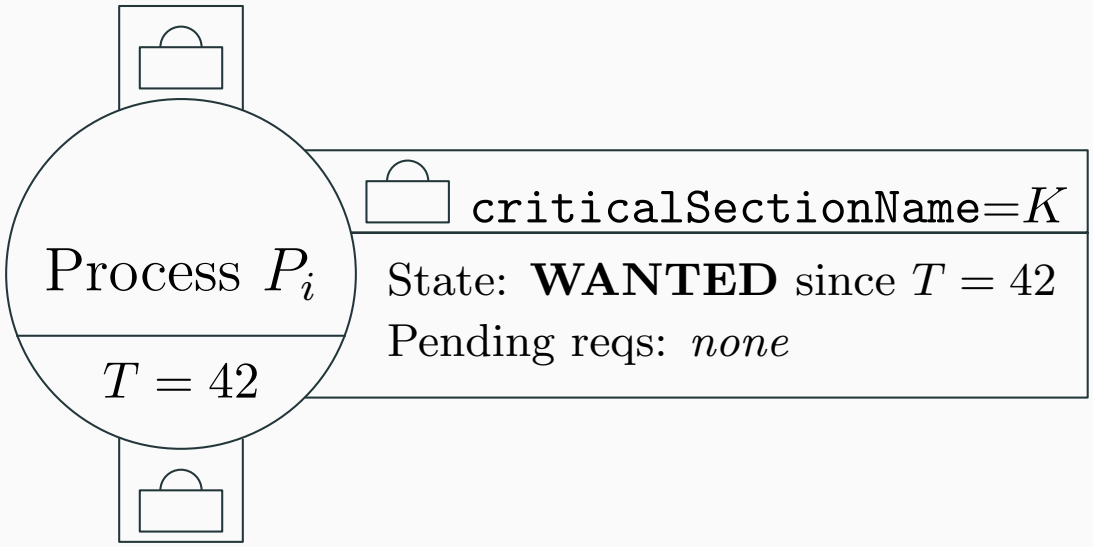


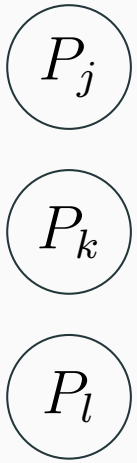
Žádost o vstup do kritické sekce



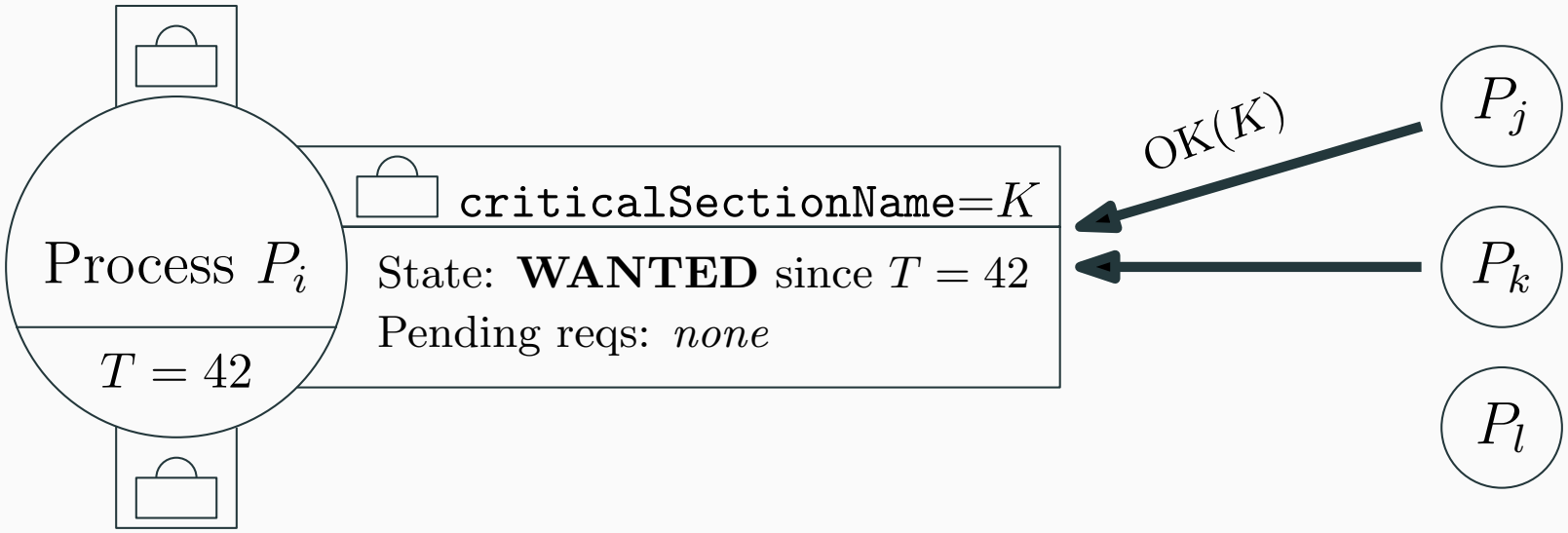
1. Pokud chce proces P_i požádat o vstup do kritické sekce K, zaznamená čas T_i kdy o zdroj žádá a pošle zprávu REQUEST(K) s tímto časem všem procesům, které do K přistupují. Nastaví stav zámku na **WANTED**.

Žádost o vstup do kritické sekce

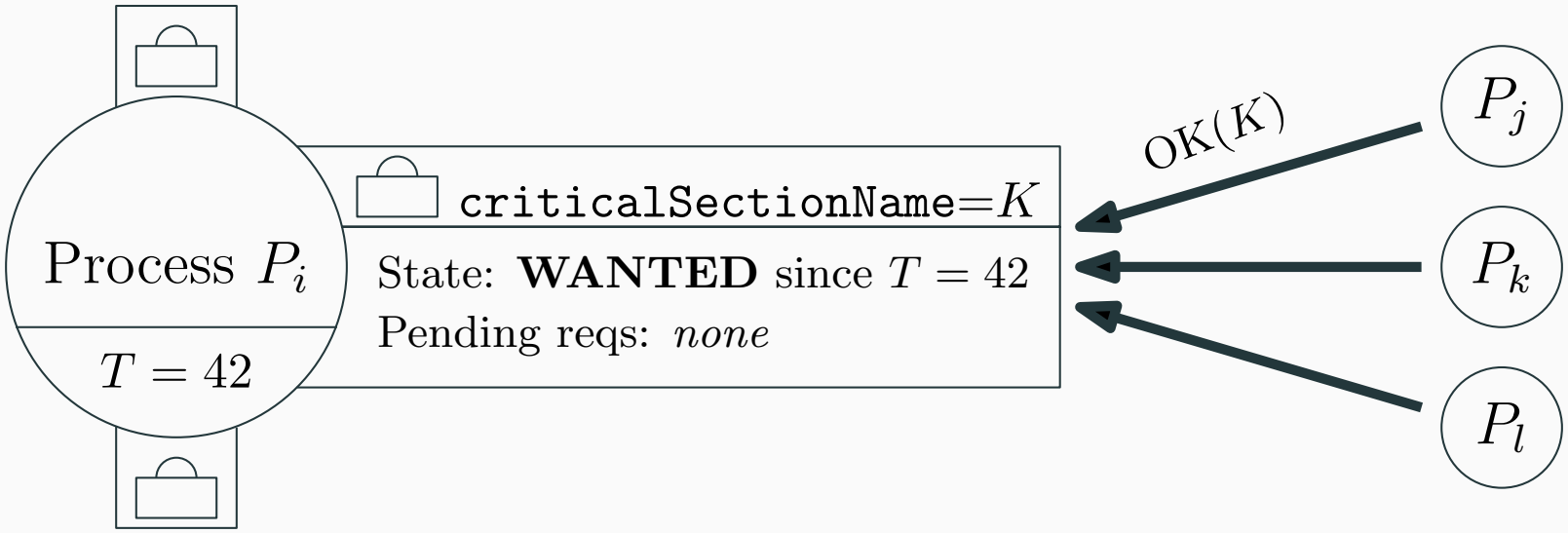




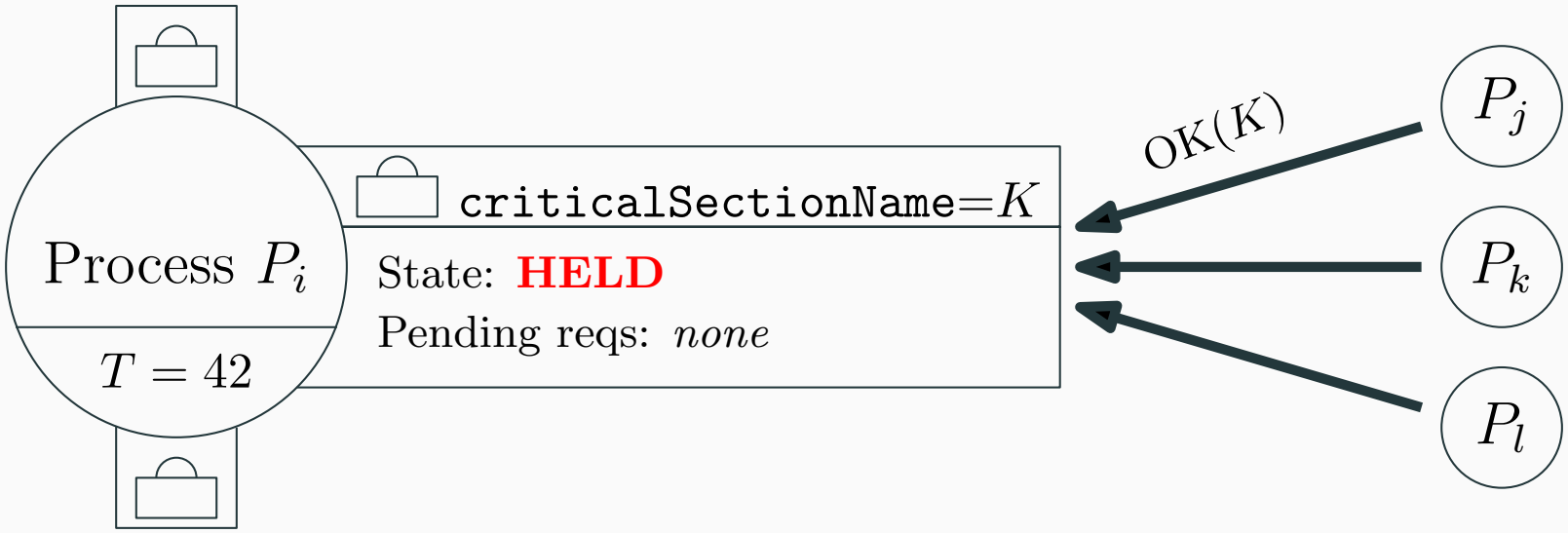
Žádost o vstup do kritické sekce



Žádost o vstup do kritické sekce



Žádost o vstup do kritické sekce



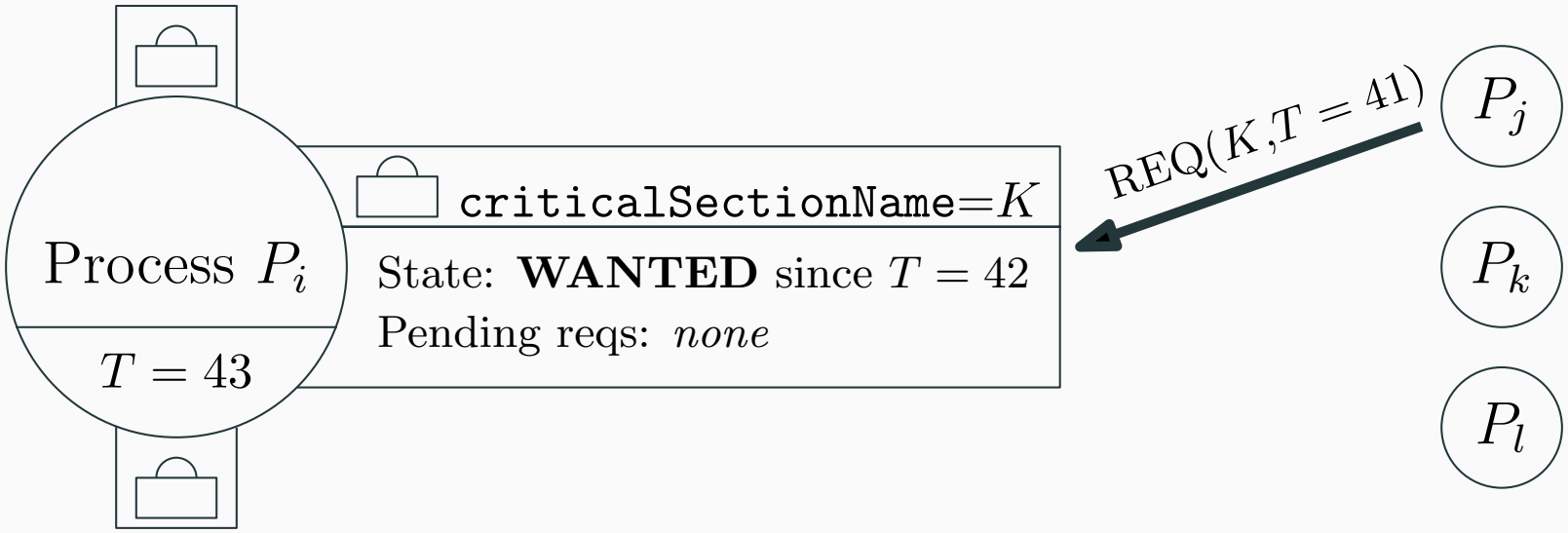
---

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

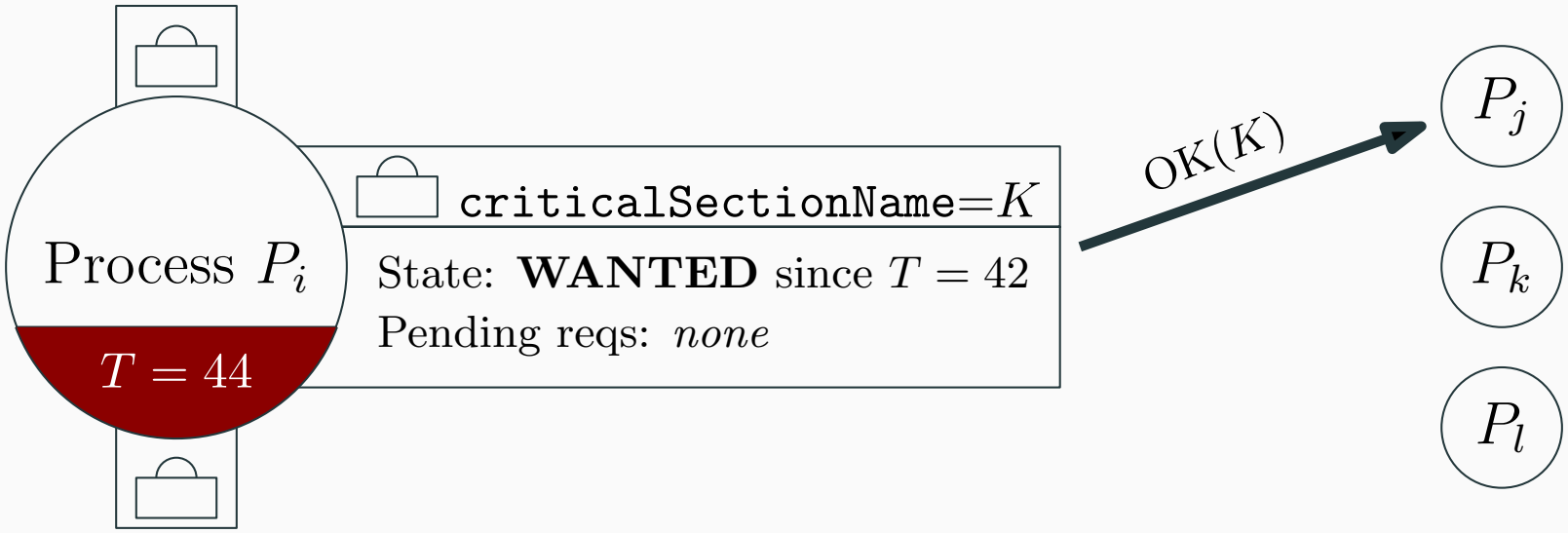
Příchozí požadavek od jiného procesu

Příchozí požadavek od jiného procesu

Příchozí požadavek od jiného procesu



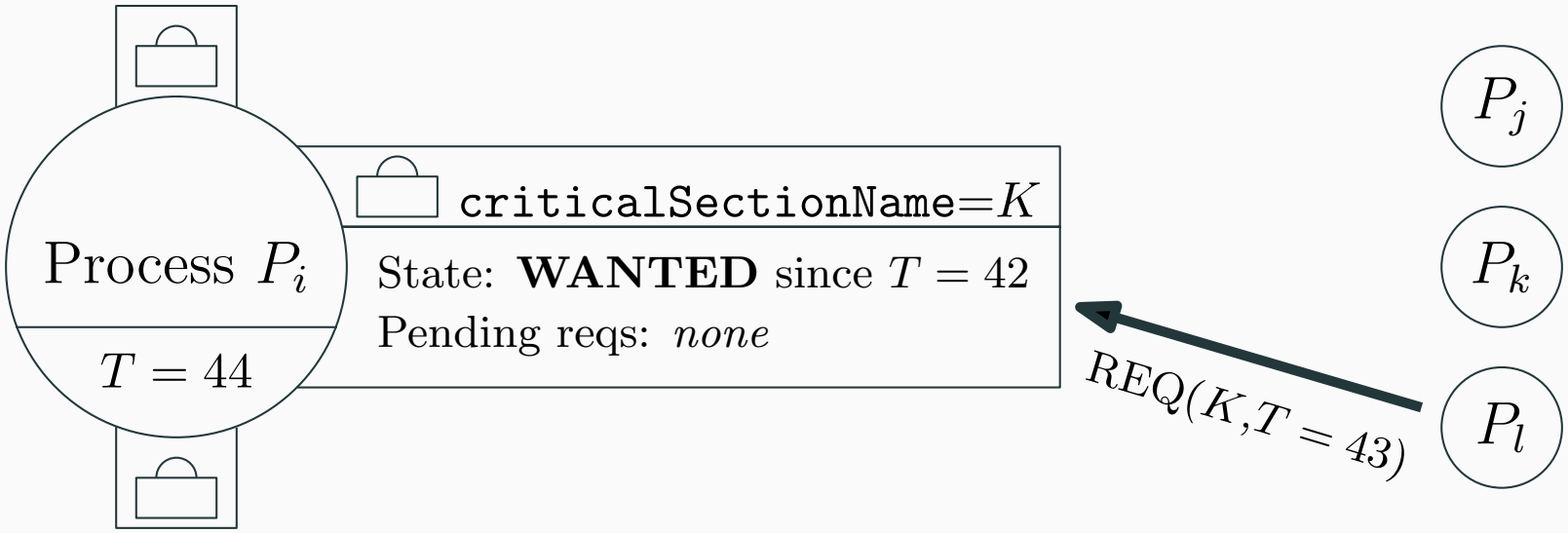
Příchozí požadavek od jiného procesu



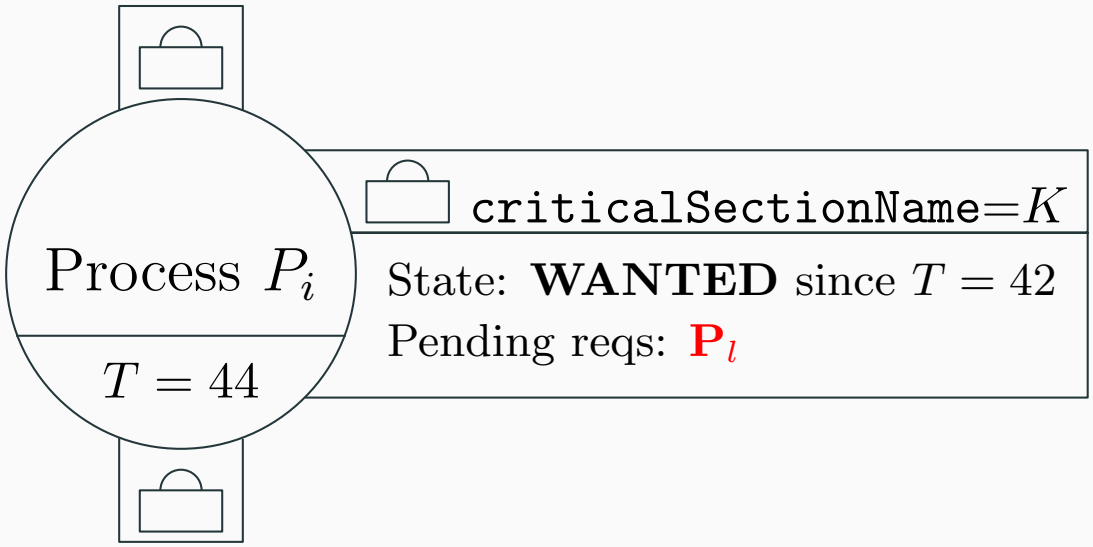
---

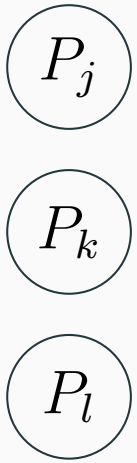
3. Pokud procesu P_i přijde zpráva REQUEST(K) od procesu P_j s časem T_j:
   (i) pokud je zámek K ve stavu **RELEASED**, nebo je ve stavu **WANTED** a o vstup do kritické sekce žádal v čase $T_i > T_j$, pak pošle zprávu OK(K) procesu P_j,

Příchozí požadavek od jiného procesu

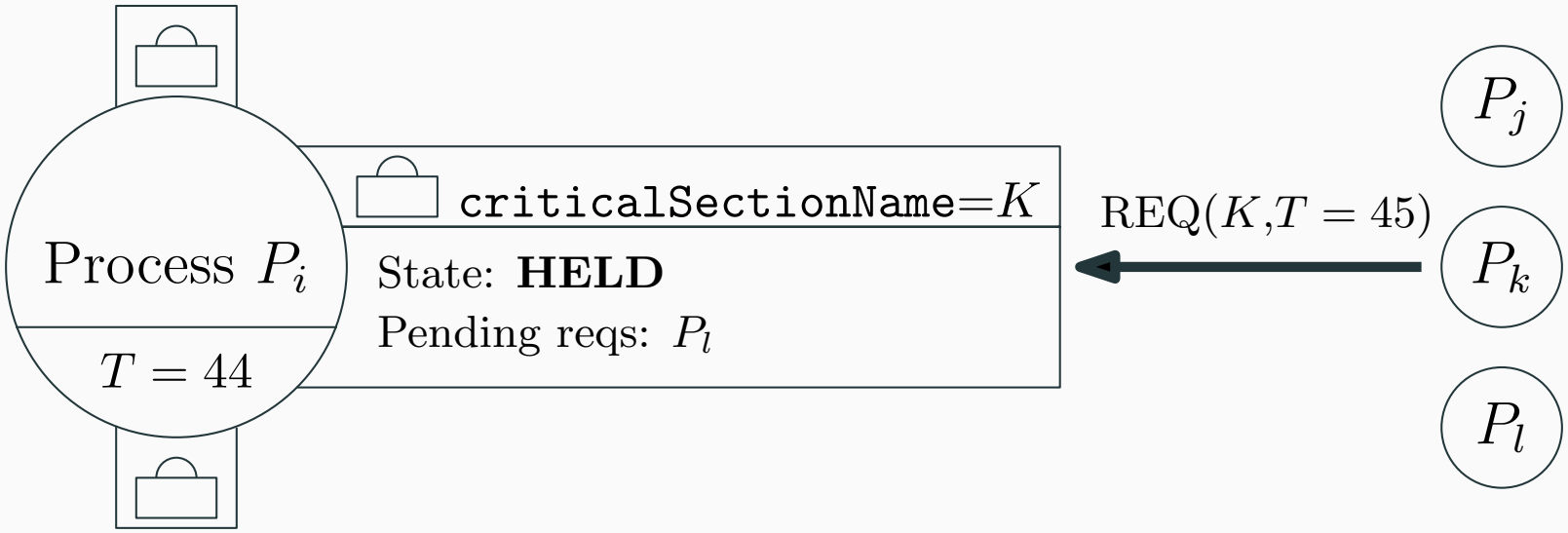


Příchozí požadavek od jiného procesu

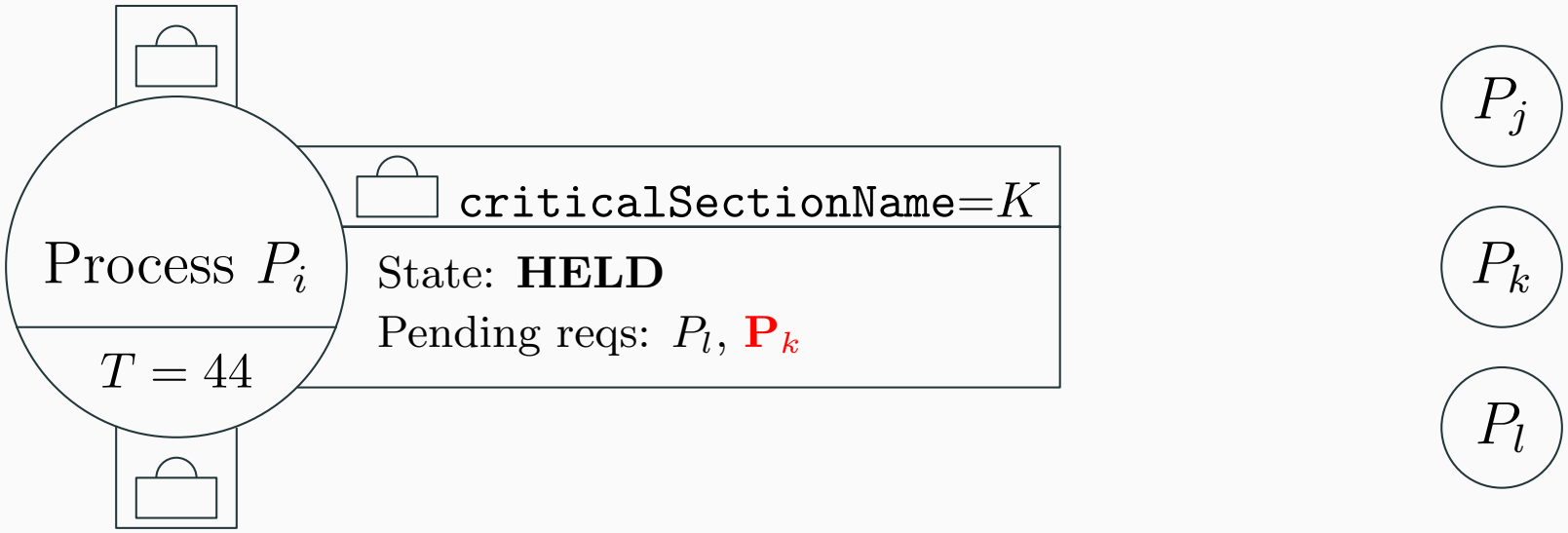




Příchozí požadavek od jiného procesu



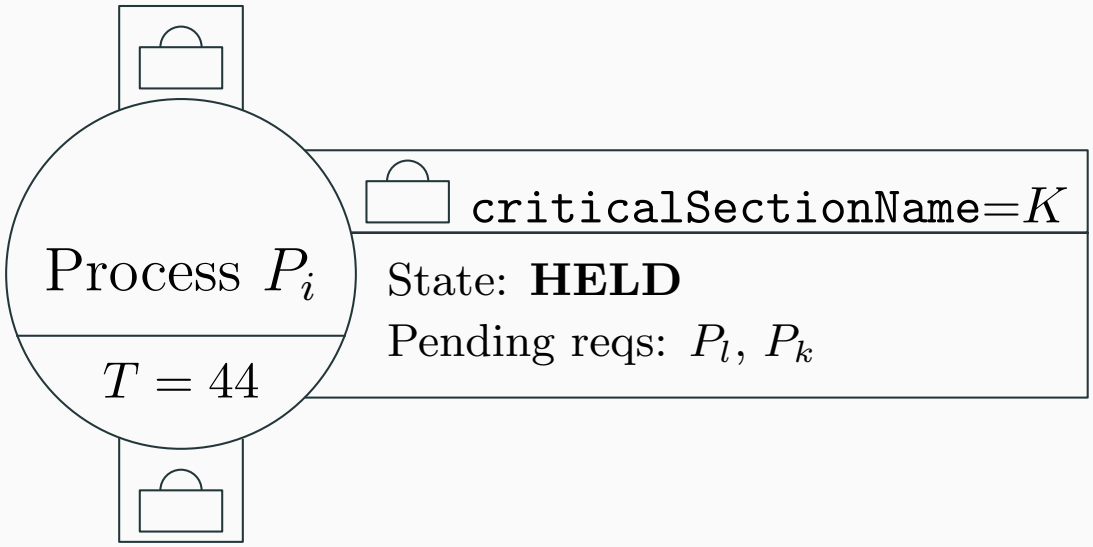
Příchozí požadavek od jiného procesu

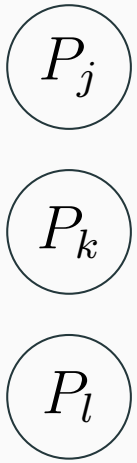


---

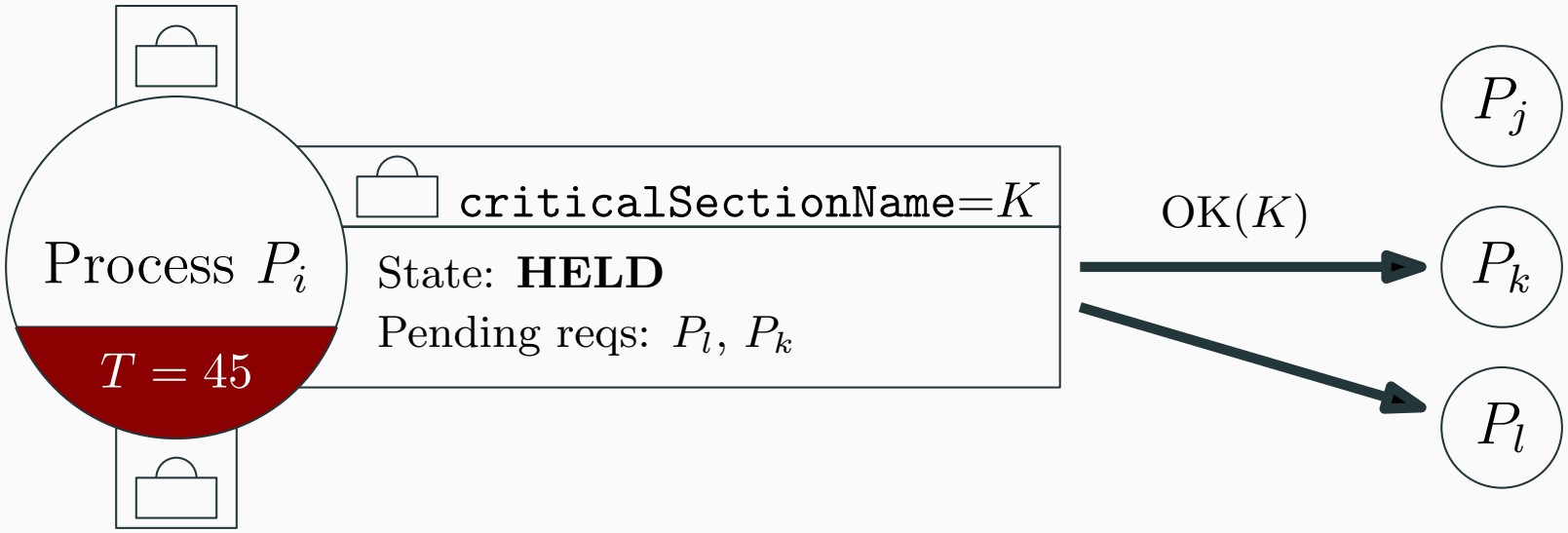
3. Pokud procesu P_i přijde zpráva REQUEST(K) od procesu P_j s časem T_j:
   (ii) jinak požadavek odloží a neodpoví.

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?

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?

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`.

⚠ **Na implementaci tentokrát pracujte již na cvičení samostatně!**

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

Termín odevzdání je **10.5. 23:59 CET** pro středeční cvičení a
11.5. 23:59 CET pro čtvrteční cvičení.

Ricart-Agrawalovo vyloučení

Každý proces má své lokální skalární logické hodiny, a zámek (či několik zámků), kde každý má

1. identifikátor kritické sekce, kterou zamyká,
2. stav: **RELEASED**, **HELD** nebo **WANTED**, a
3. frontu odložených požadavků.

V systému kolují pro každou kritickou sekci dva typy zpráv: **REQUEST** a **OK**

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**.
2. Zámek K procesu je ve stavu **WANTED** dokud neobdrží zprávu OK(K) od každého dalšího přistupujícího procesu. Poté se nastaví na **HELD**.
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 $T_i > T_j$, případně $T_i = T_j$ a $i > j$, pak pošle zprávu OK(K) procesu P_j,
   (ii) jinak požadavek odloží a neodpoví.
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.

Díky za pozornost!

Budeme rádi za Vaši zpětnou vazbu! $\rightarrow$

https://goo.gl/forms/URUxIrHCH2TsTFi92