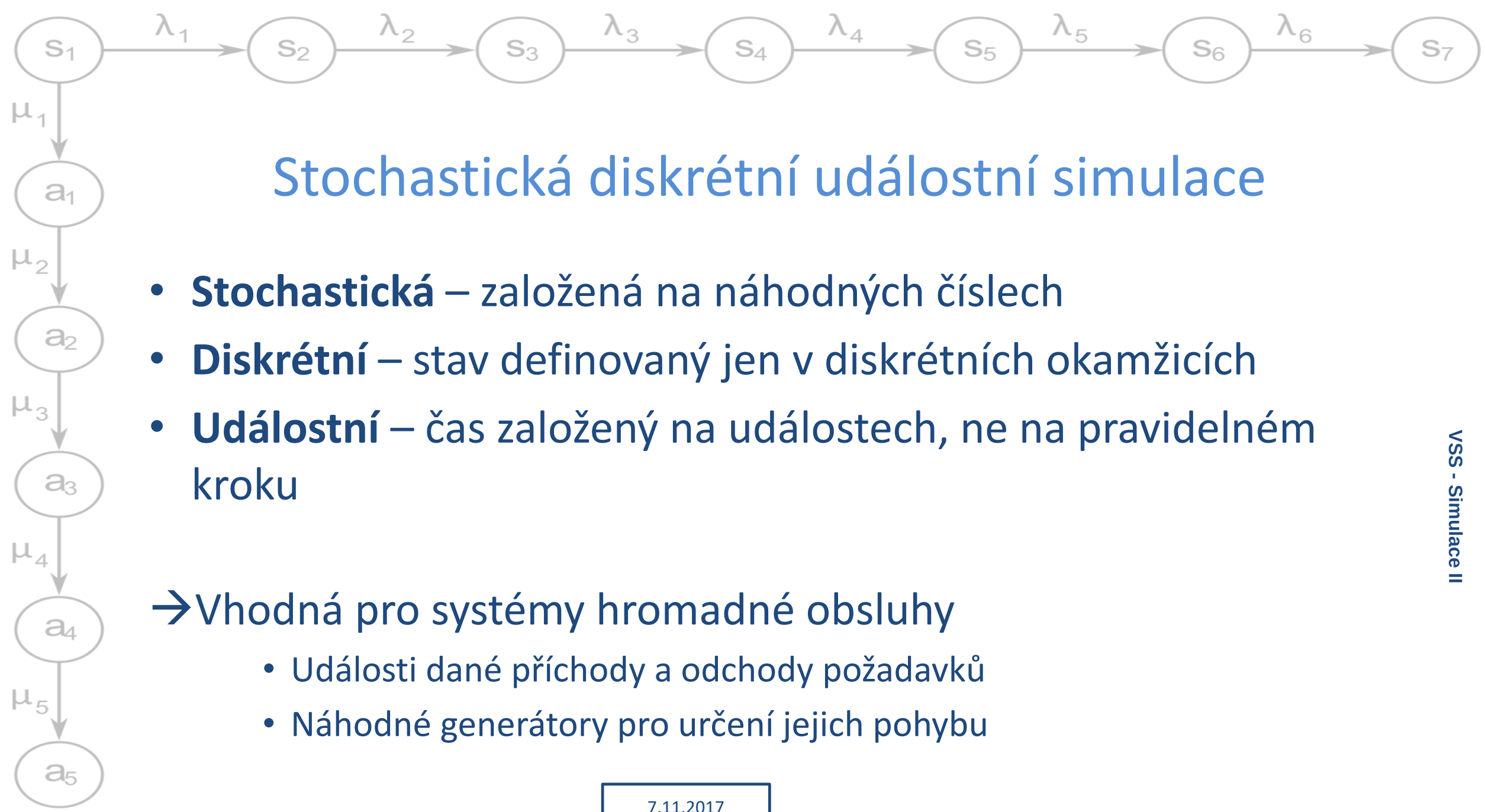
Implementace simulačního modelu

Využití JSimu a obecné rady k implementaci diskrétní událostní simulace

Richard Lipka

7.11. 2017

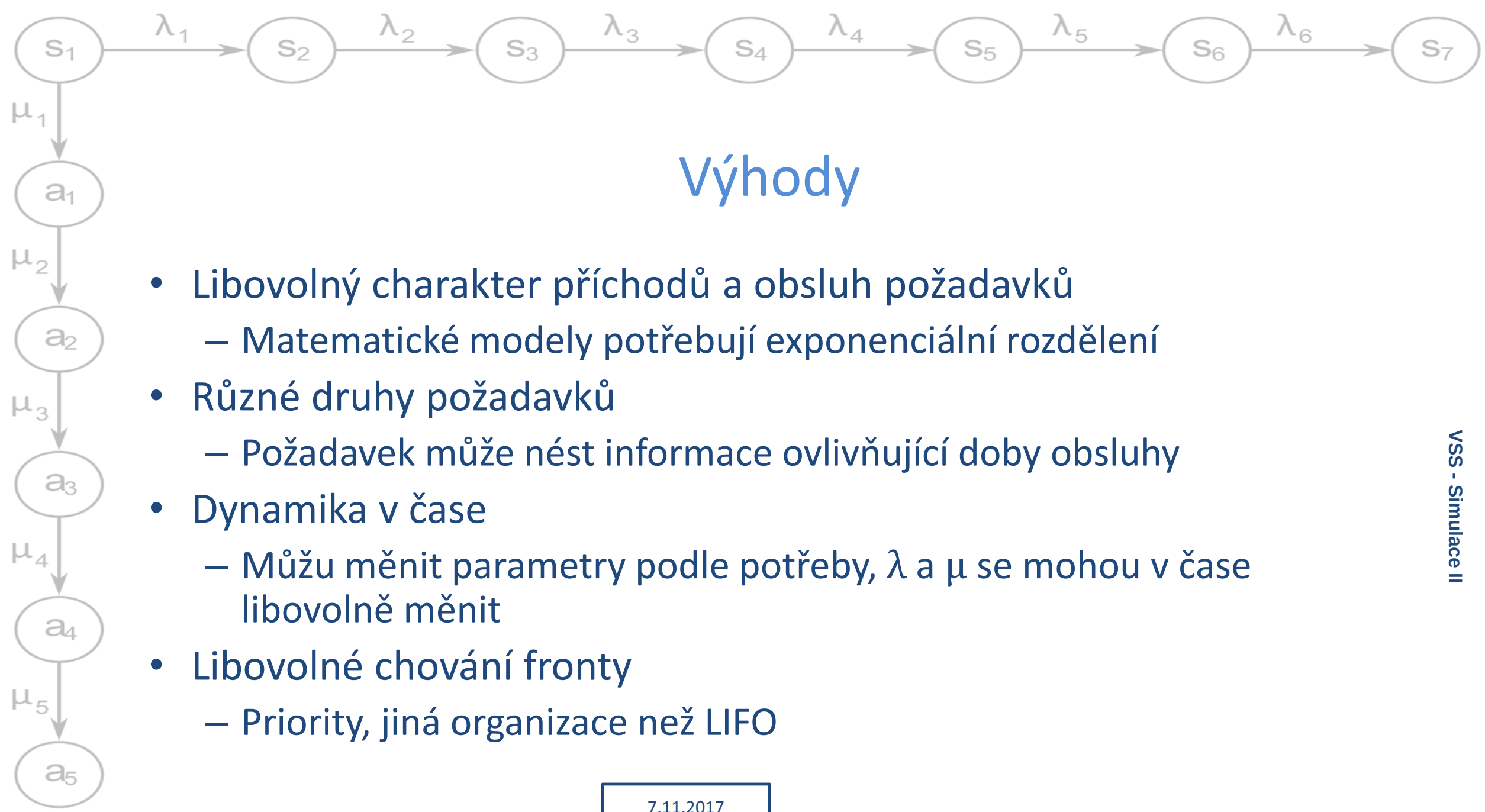


Stochastická diskrétní událostní simulace

- **Stochastická** – založená na náhodných číslech
- **Diskrétní** – stav definovaný jen v diskrétních okamžicích
- **Událostní** – čas založený na událostech, ne na pravidelném kroku

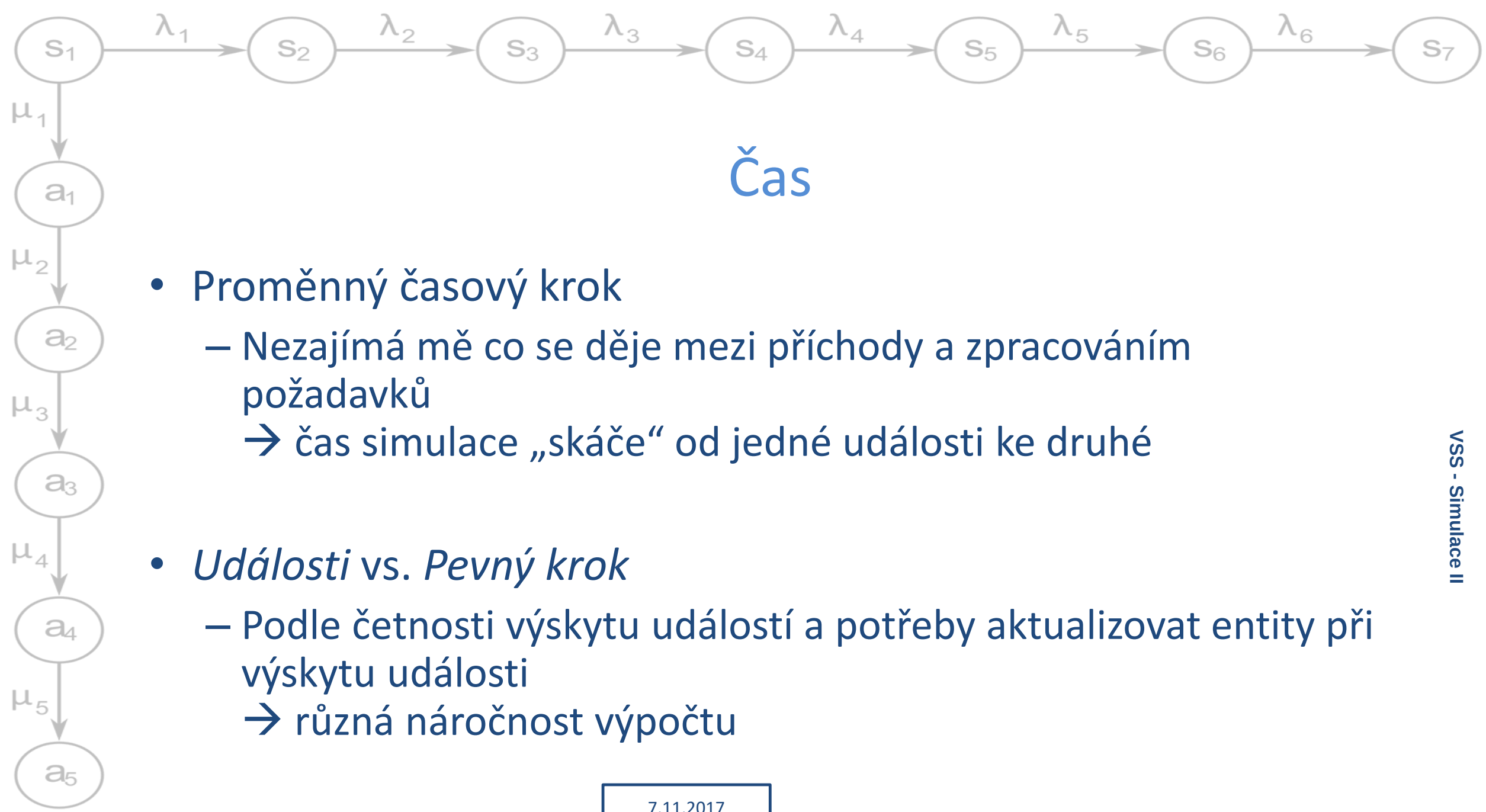
→Vhodná pro systémy hromadné obsluhy

- Události dané příchody a odchody požadavků
- Náhodné generátory pro určení jejich pohybu



Výhody

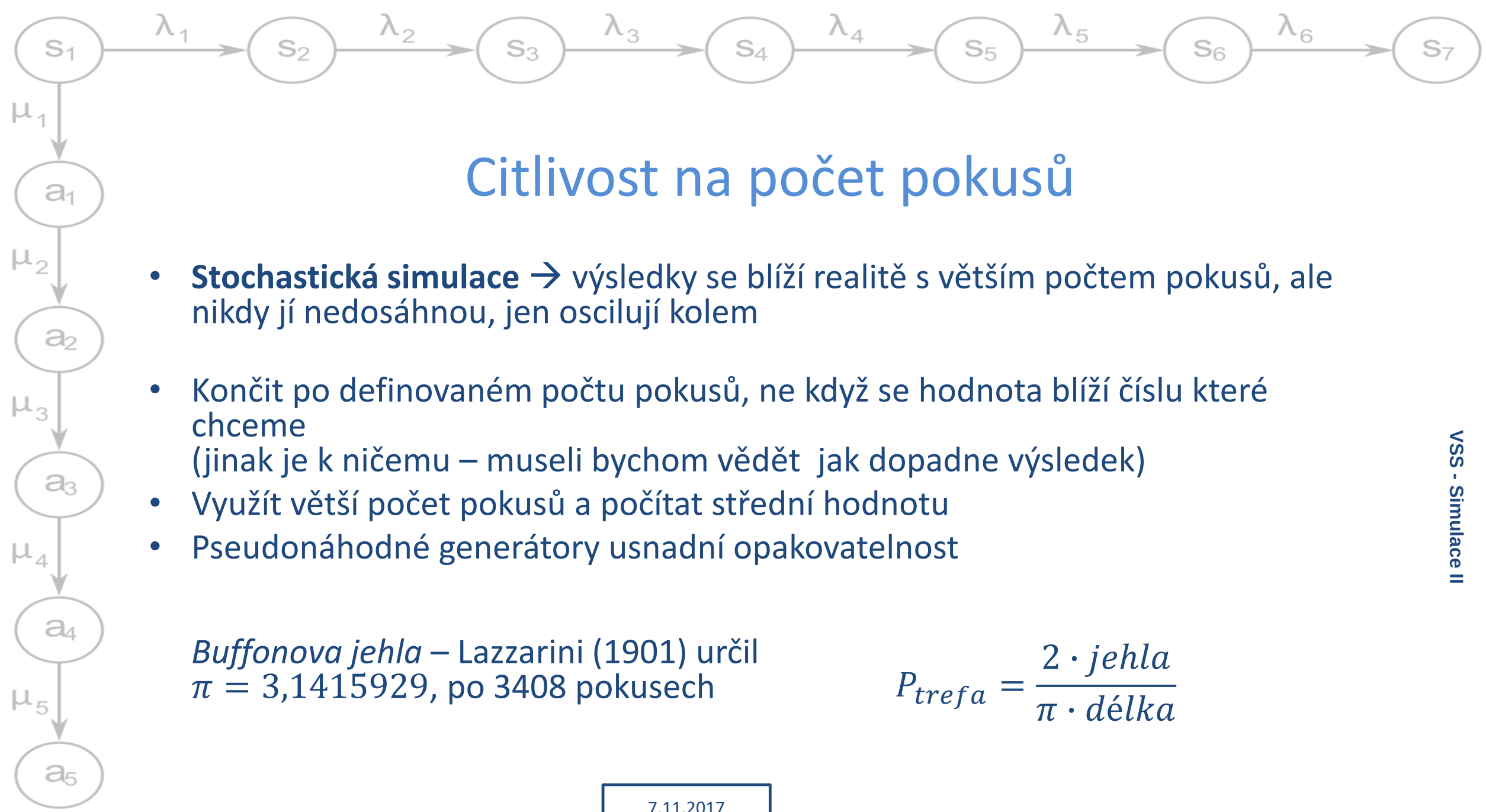
- Libovolný charakter příchodů a obsluh požadavků
  - Matematické modely potřebují exponenciální rozdělení
- Různé druhy požadavků
  - Požadavek může nést informace ovlivňující doby obsluhy
- Dynamika v čase
  - Můžu měnit parametry podle potřeby, λ a μ se mohou v čase libovolně měnit
- Libovolné chování fronty
  - Priority, jiná organizace než LIFO



Čas

- Proměnný časový krok
  - Nezajímá mě co se děje mezi příchody a zpracováním požadavků
    → čas simulace „skáče“ od jedné události ke druhé

- *Události* vs. *Pevný krok*
  - Podle četnosti výskytu událostí a potřeby aktualizovat entity při výskytu události
    → různá náročnost výpočtu



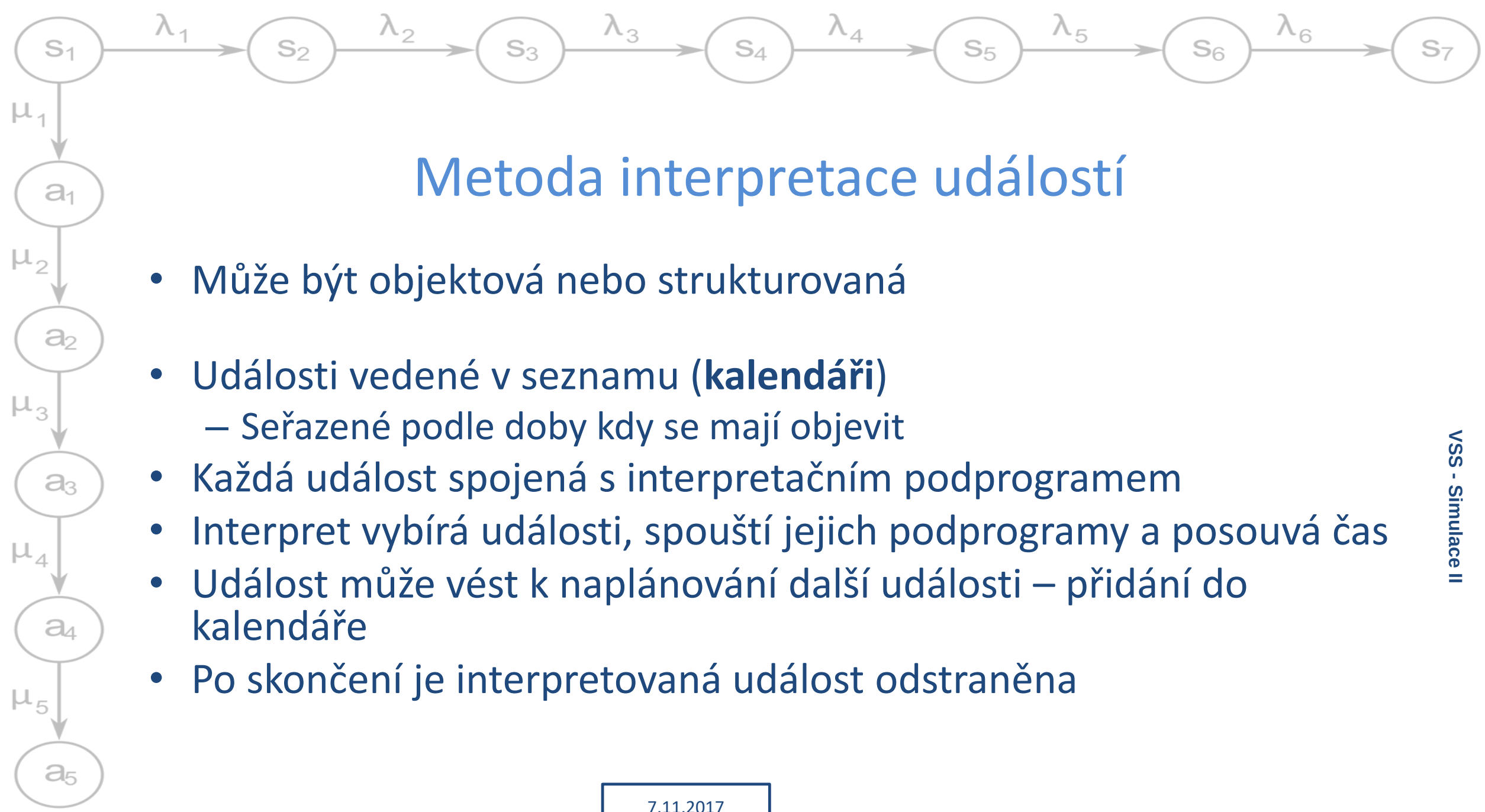
Citlivost na počet pokusů

- **Stochastická simulace** → výsledky se blíží realitě s větším počtem pokusů, ale nikdy jí nedosáhnou, jen oscilují kolem

- Končit po definovaném počtu pokusů, ne když se hodnota blíží číslu které chceme
  (jinak je k ničemu – museli bychom vědět jak dopadne výsledek)
- Využít větší počet pokusů a počítat střední hodnotu
- Pseudonáhodné generátory usnadní opakovatelnost

Buffonova jehla – Lazzarini (1901) určil $\pi = 3{,}1415929$, po 3408 pokusech

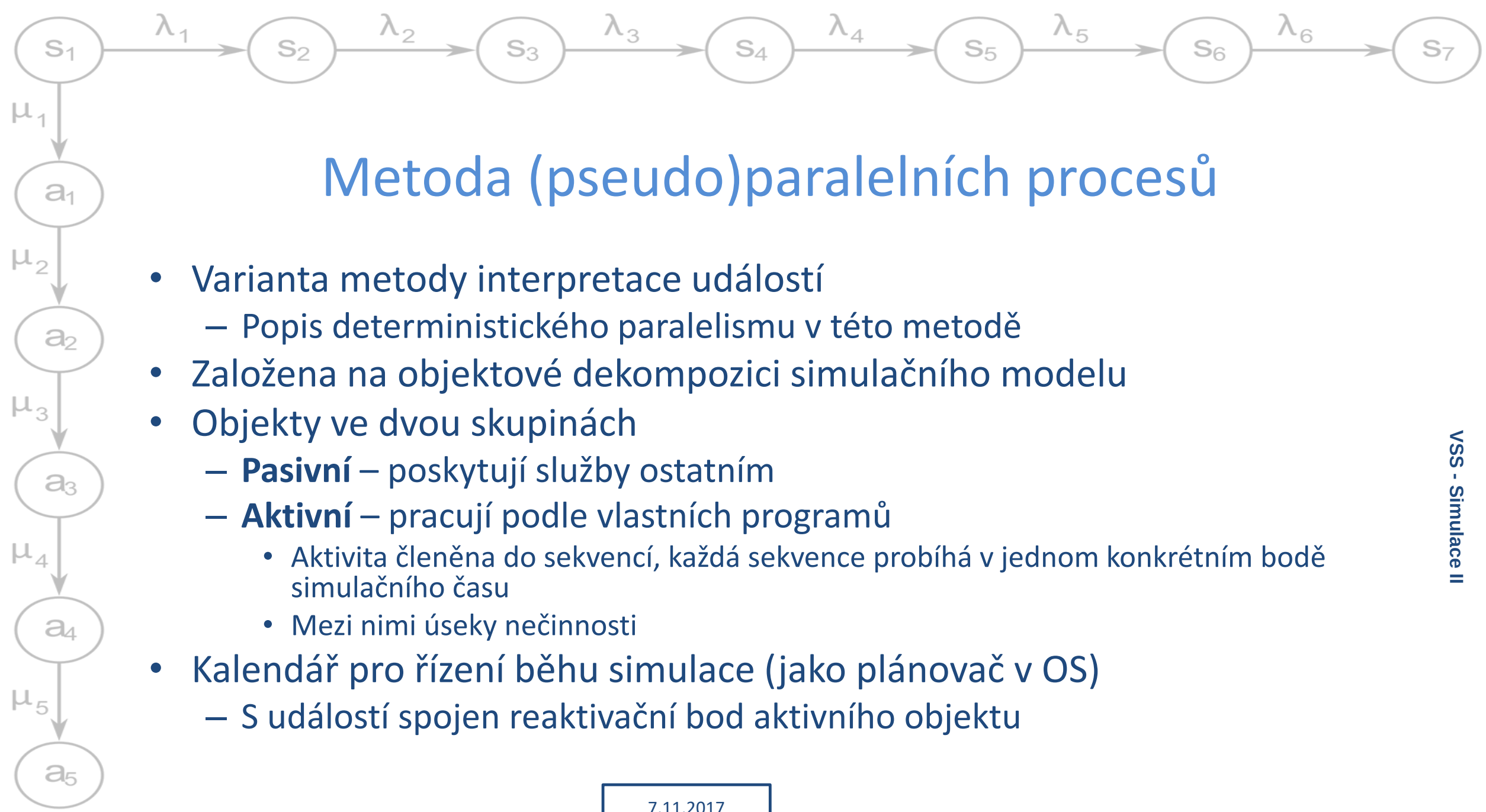
$$P_{trefa} = \frac{2 \cdot jehla}{\pi \cdot délka}$$



Metoda interpretace událostí

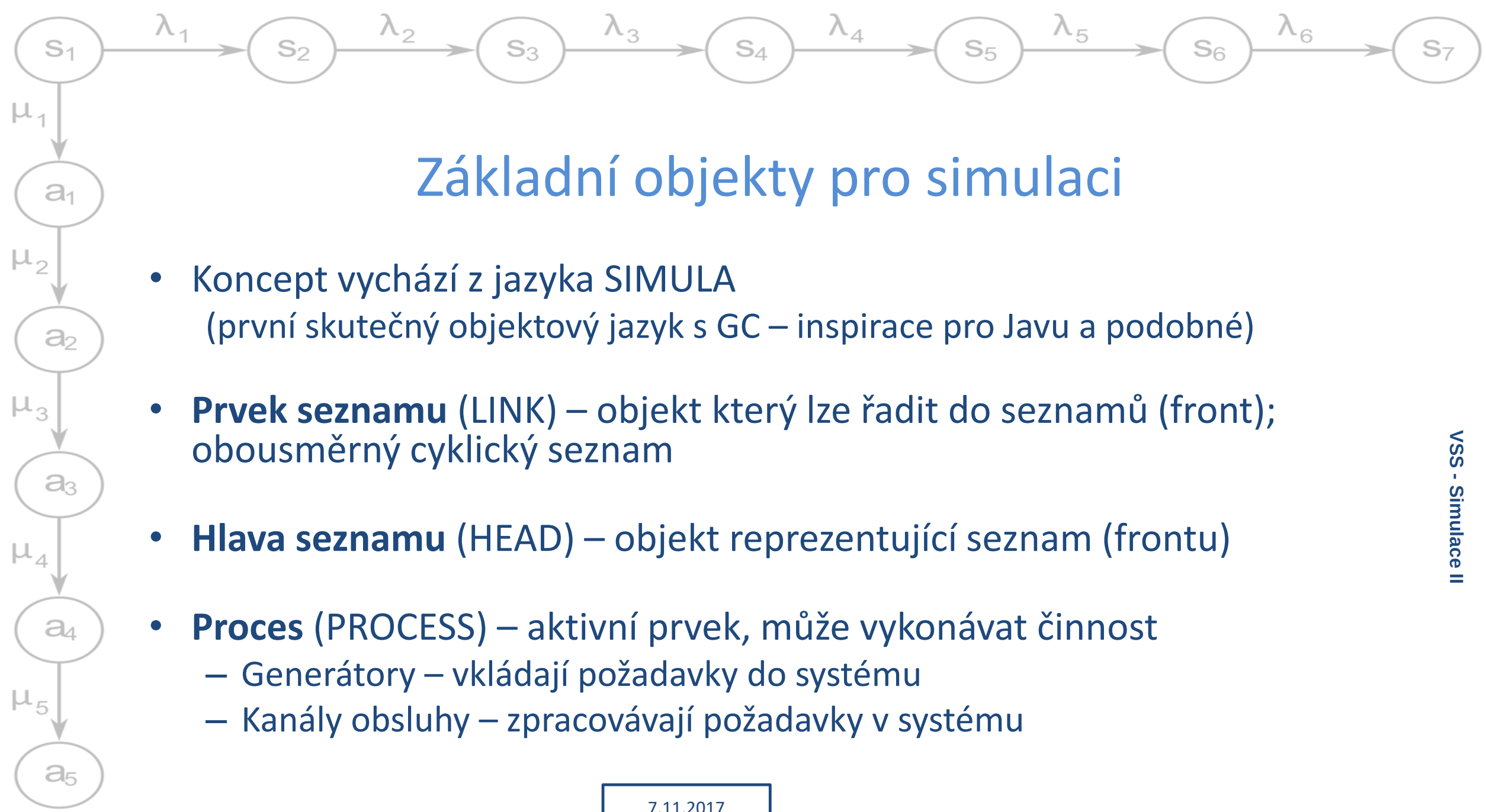
- Může být objektová nebo strukturovaná

- Události vedené v seznamu (**kalendáři**)
  - Seřazené podle doby kdy se mají objevit
- Každá událost spojená s interpretačním podprogramem
- Interpret vybírá události, spouští jejich podprogramy a posouvá čas
- Událost může vést k naplánování další události – přidání do kalendáře
- Po skončení je interpretovaná událost odstraněna



Metoda (pseudo)paralelních procesů

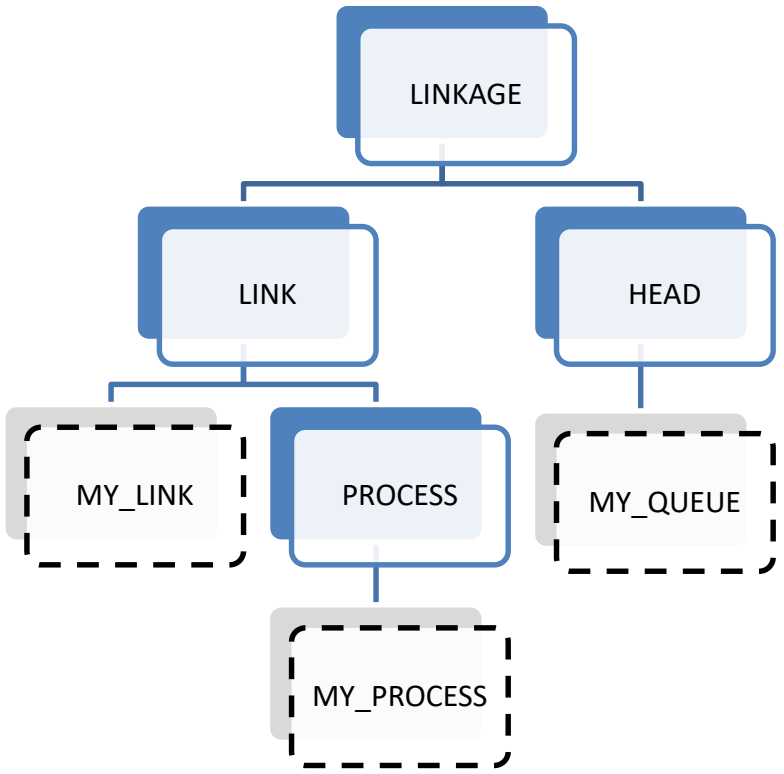
- Varianta metody interpretace událostí
  - Popis deterministického paralelismu v této metodě
- Založena na objektové dekompozici simulačního modelu
- Objekty ve dvou skupinách
  - **Pasivní** – poskytují služby ostatním
  - **Aktivní** – pracují podle vlastních programů
    - Aktivita členěna do sekvencí, každá sekvence probíhá v jednom konkrétním bodě simulačního času
    - Mezi nimi úseky nečinnosti
- Kalendář pro řízení běhu simulace (jako plánovač v OS)
  - S událostí spojen reaktivační bod aktivního objektu



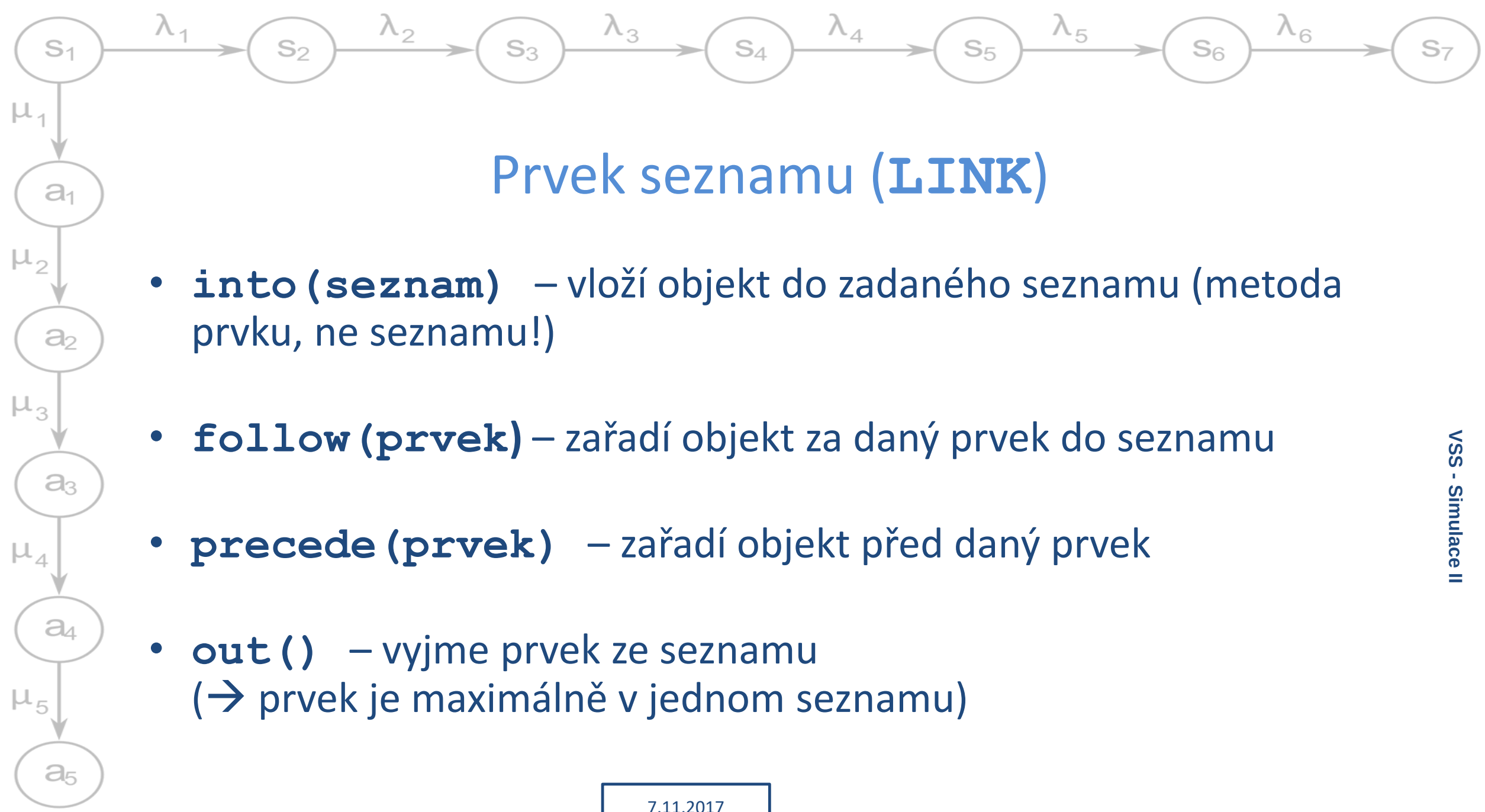
Základní objekty pro simulaci

- Koncept vychází z jazyka SIMULA
  (první skutečný objektový jazyk s GC – inspirace pro Javu a podobné)
- **Prvek seznamu** (LINK) – objekt který lze řadit do seznamů (front); obousměrný cyklický seznam
- **Hlava seznamu** (HEAD) – objekt reprezentující seznam (frontu)
- **Proces** (PROCESS) – aktivní prvek, může vykonávat činnost
  - Generátory – vkládají požadavky do systému
  - Kanály obsluhy – zpracovávají požadavky v systému

Struktura objektů

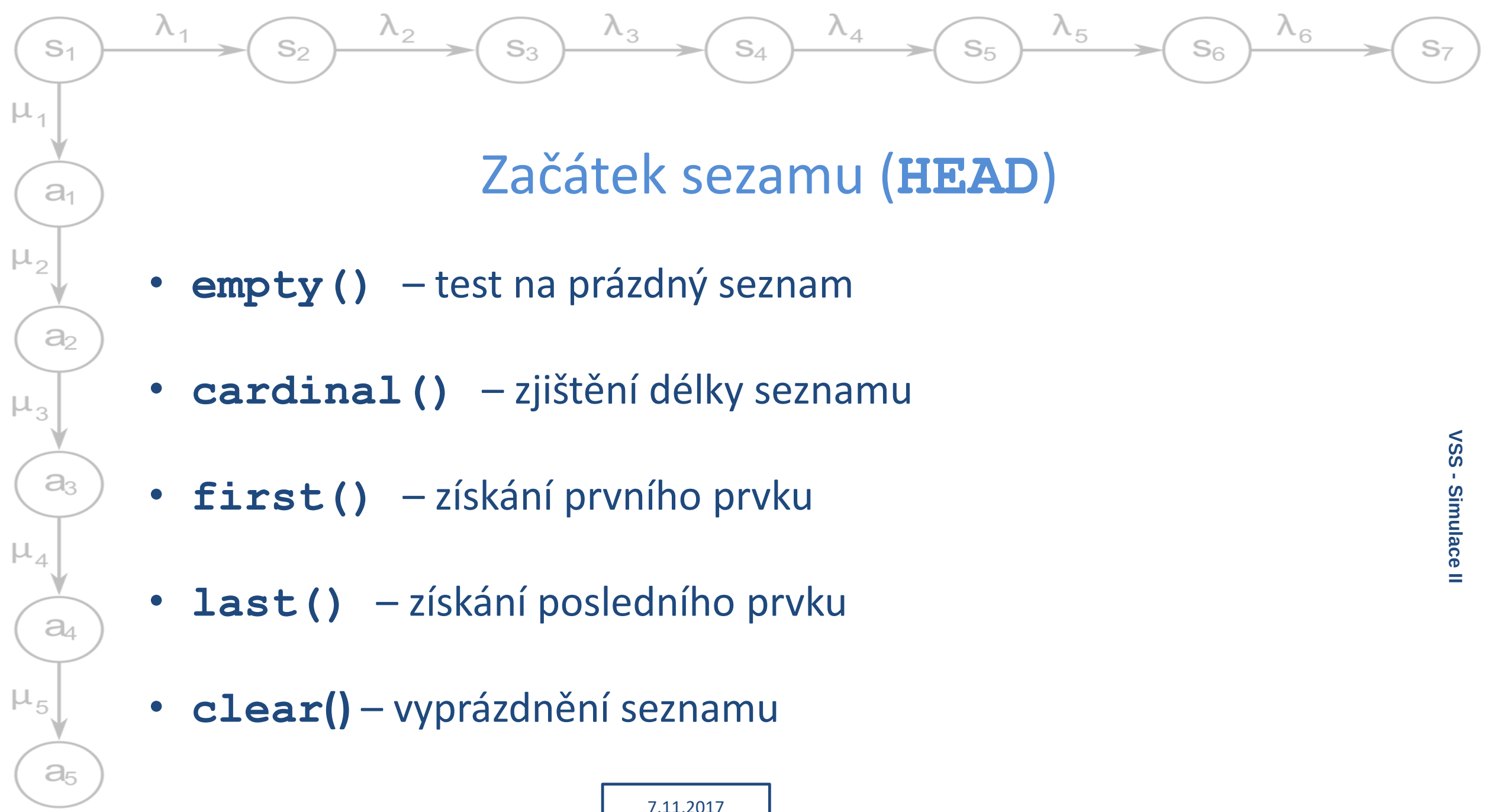


- Procesy je možné řadit do front
  - Kalendář lze chápat jako speciální frontu



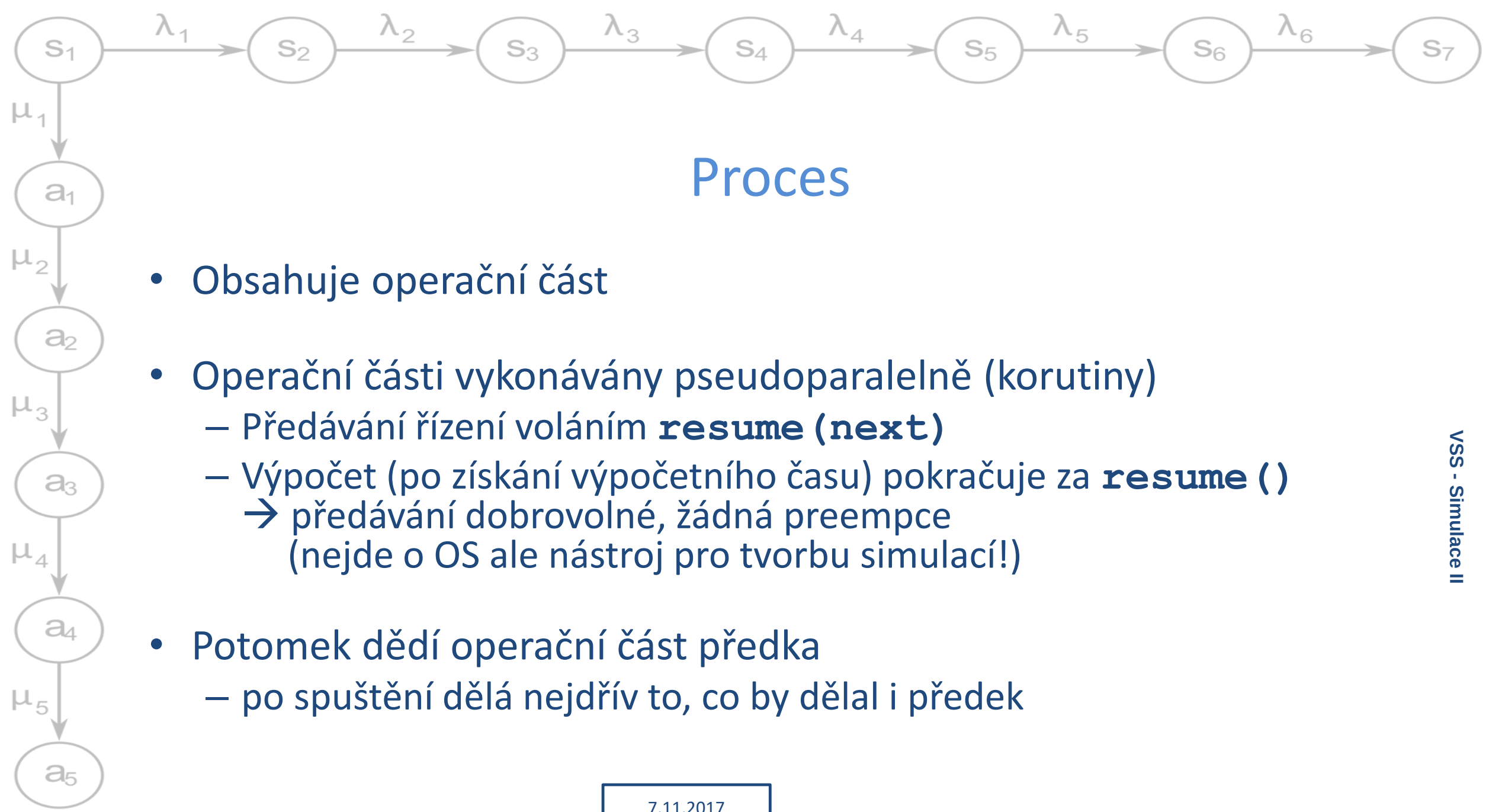
Prvek seznamu (**`LINK`**)

- **`into(seznam)`** – vloží objekt do zadaného seznamu (metoda prvku, ne seznamu!)
- **`follow(prvek)`** – zařadí objekt za daný prvek do seznamu
- **`precede(prvek)`** – zařadí objekt před daný prvek
- **`out()`** – vyjme prvek ze seznamu
  (→ prvek je maximálně v jednom seznamu)



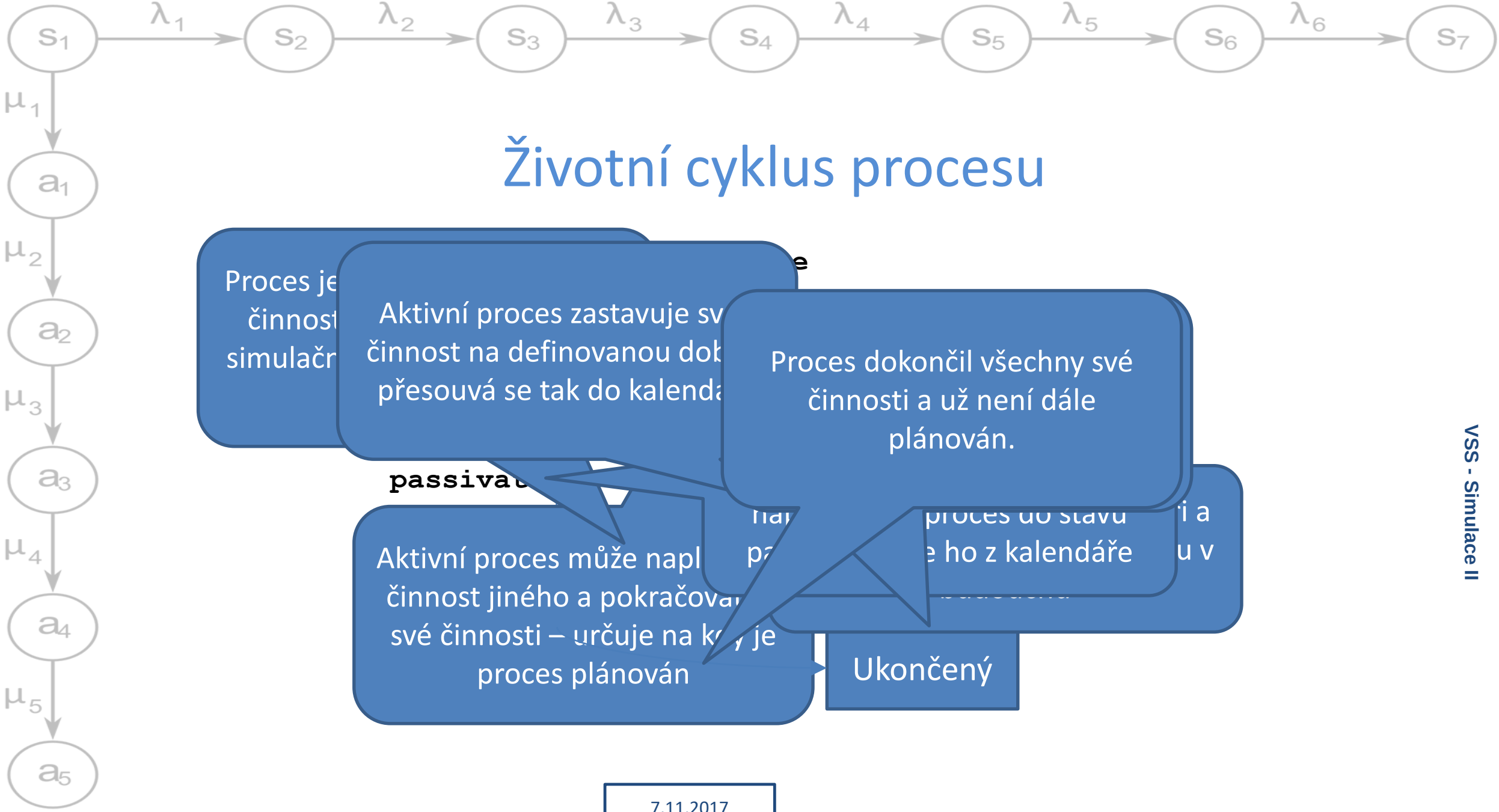
Začátek sezamu (**HEAD**)

- **`empty()`** – test na prázdný seznam
- **`cardinal()`** – zjištění délky seznamu
- **`first()`** – získání prvního prvku
- **`last()`** – získání posledního prvku
- **`clear()`** – vyprázdnění seznamu



Proces

- Obsahuje operační část
- Operační části vykonávány pseudoparalelně (korutiny)
  - Předávání řízení voláním **`resume(next)`**
  - Výpočet (po získání výpočetního času) pokračuje za **`resume()`**
    → předávání dobrovolné, žádná preempce
    (nejde o OS ale nástroj pro tvorbu simulací!)
- Potomek dědí operační část předka
  - po spuštění dělá nejdřív to, co by dělal i předek



Životní cyklus procesu

Proces je
činnost
simulačn

Aktivní proces zastavuje sv
činnost na definovanou dob
přesouvá se tak do kalenda

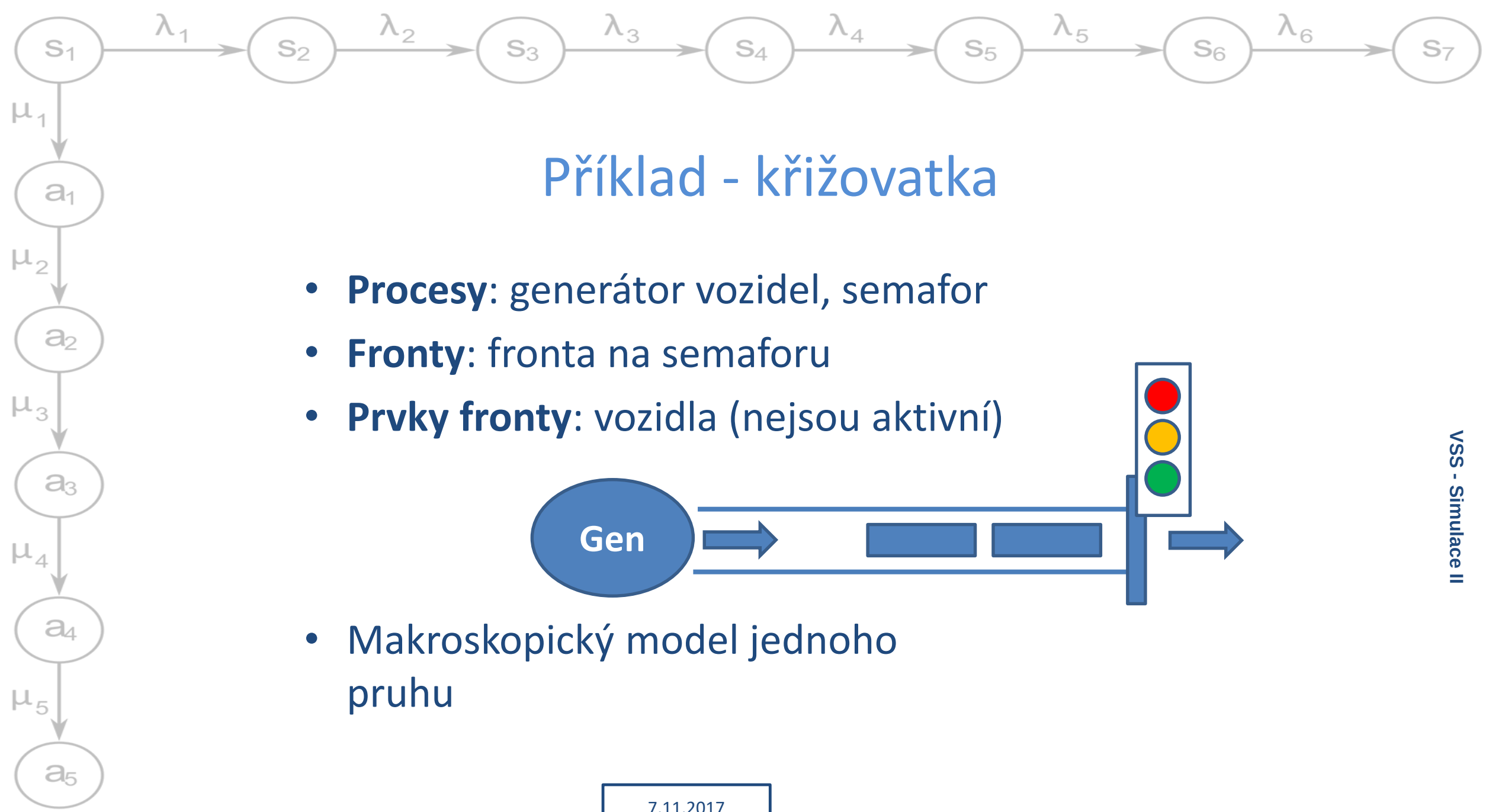
Proces dokončil všechny své činnosti a už není dále plánován.

passivat

Aktivní proces může napl
činnost jiného a pokračova
své činnosti – určuje na k
proces plánován

proces do stavu
e ho z kalendáře

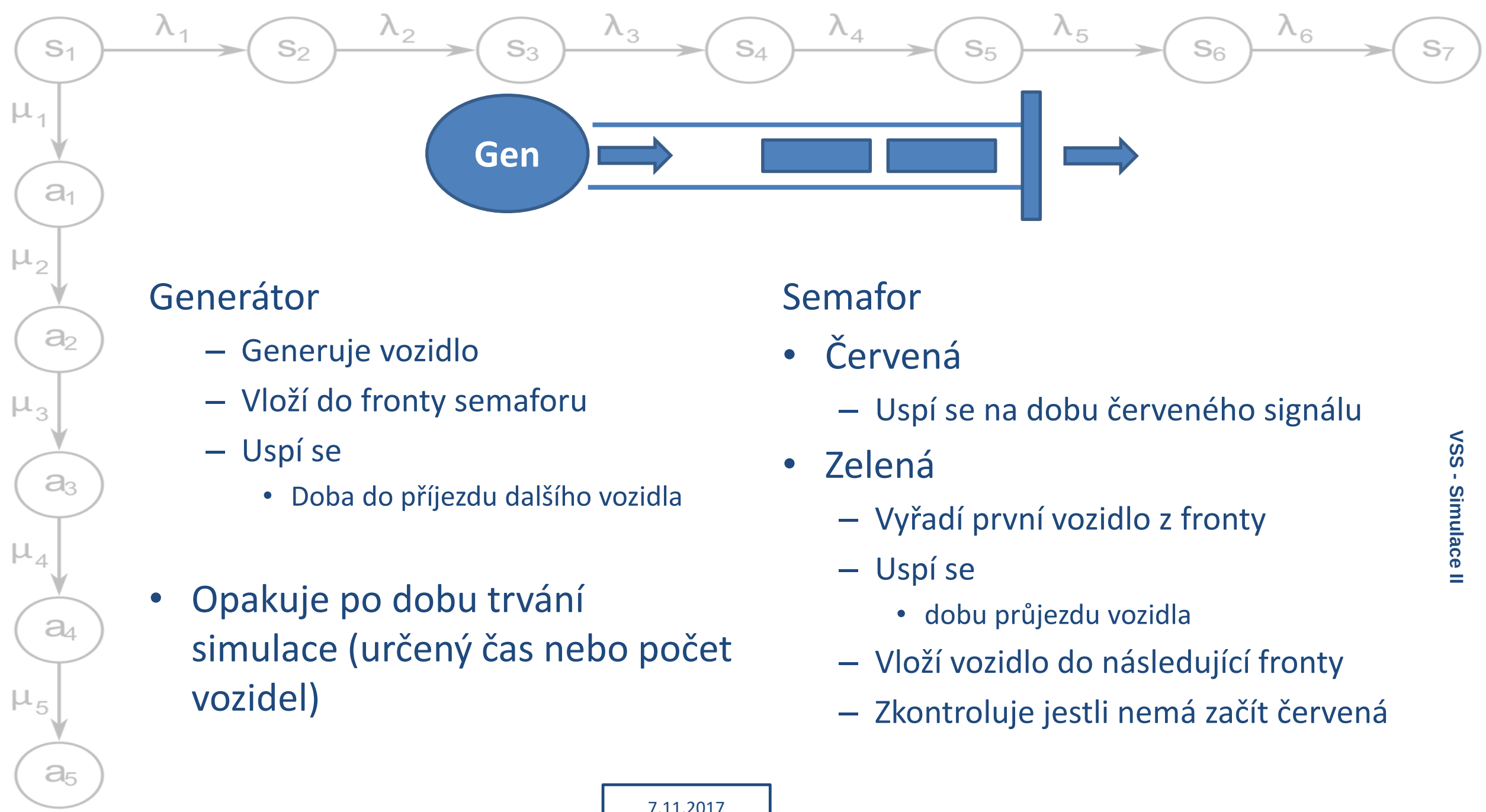
Ukončený



Příklad - křižovatka

- **Procesy**: generátor vozidel, semafor
- **Fronty**: fronta na semaforu
- **Prvky fronty**: vozidla (nejsou aktivní)

- Makroskopický model jednoho pruhu



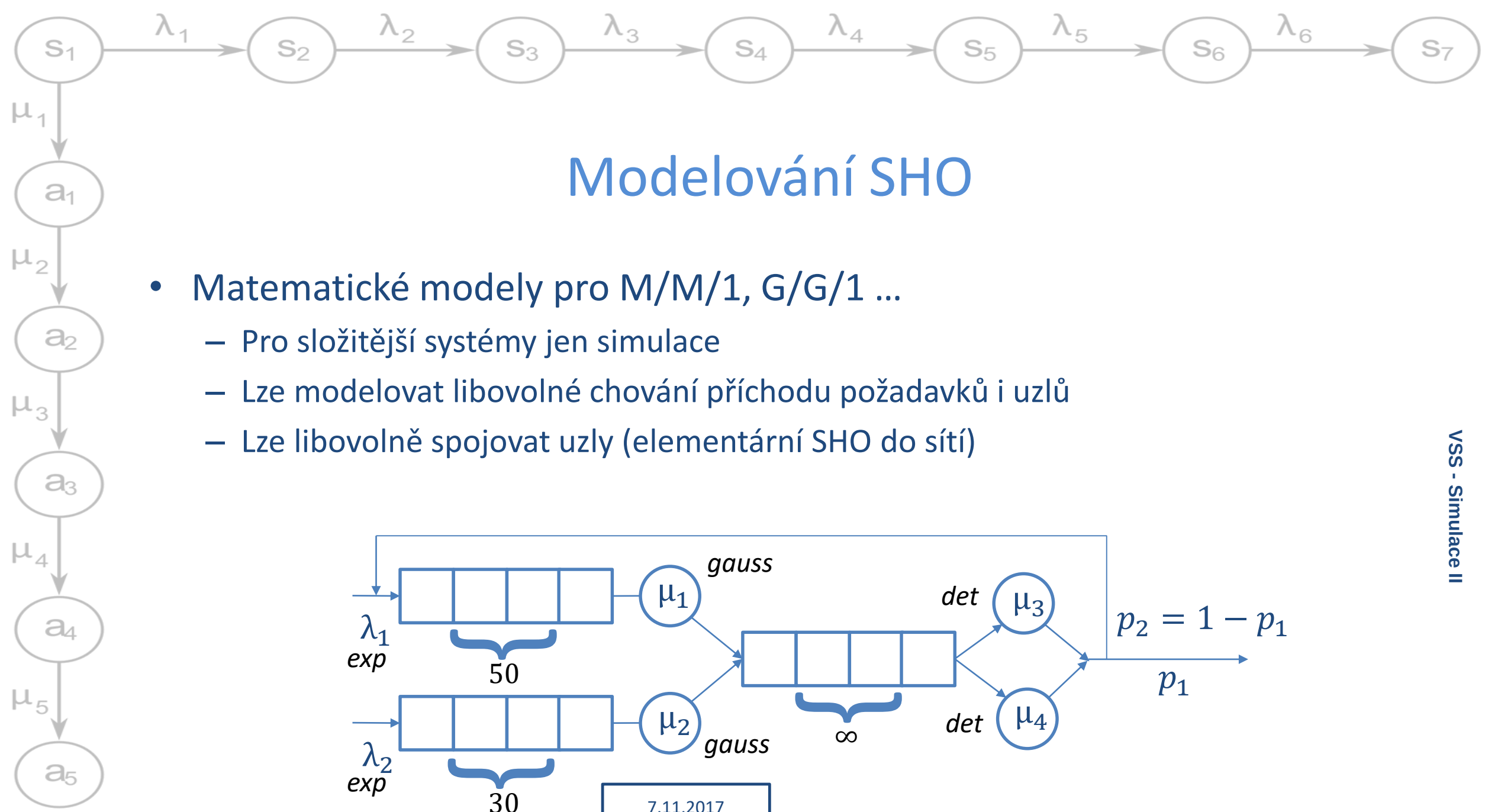
Generátor

- Generuje vozidlo
- Vloží do fronty semaforu
- Uspí se
  - Doba do příjezdu dalšího vozidla

- Opakuje po dobu trvání simulace (určený čas nebo počet vozidel)

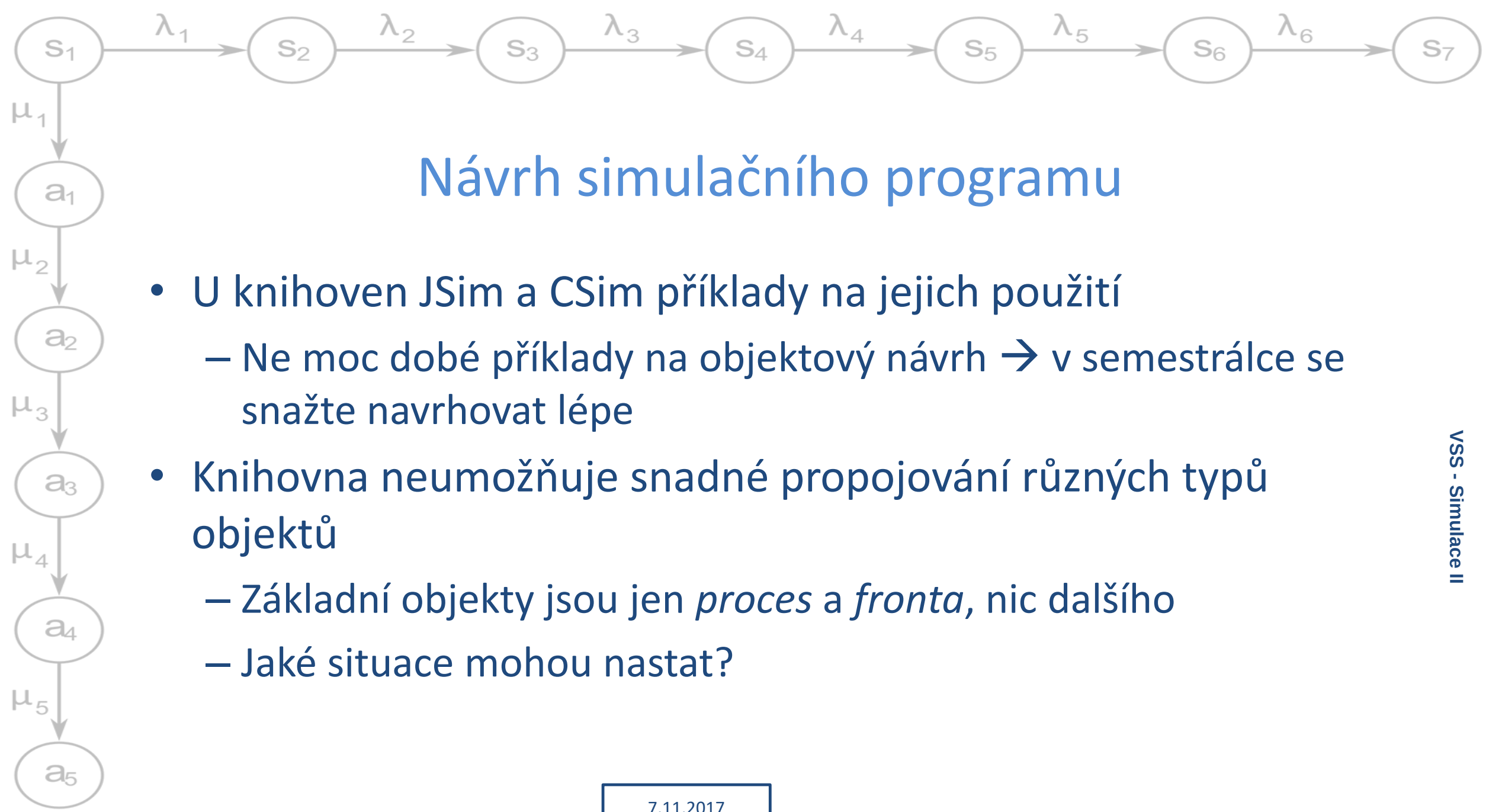
Semafor

- Červená
  - Uspí se na dobu červeného signálu
- Zelená
  - Vyřadí první vozidlo z fronty
  - Uspí se
    - dobu průjezdu vozidla
  - Vloží vozidlo do následující fronty
  - Zkontroluje jestli nemá začít červená



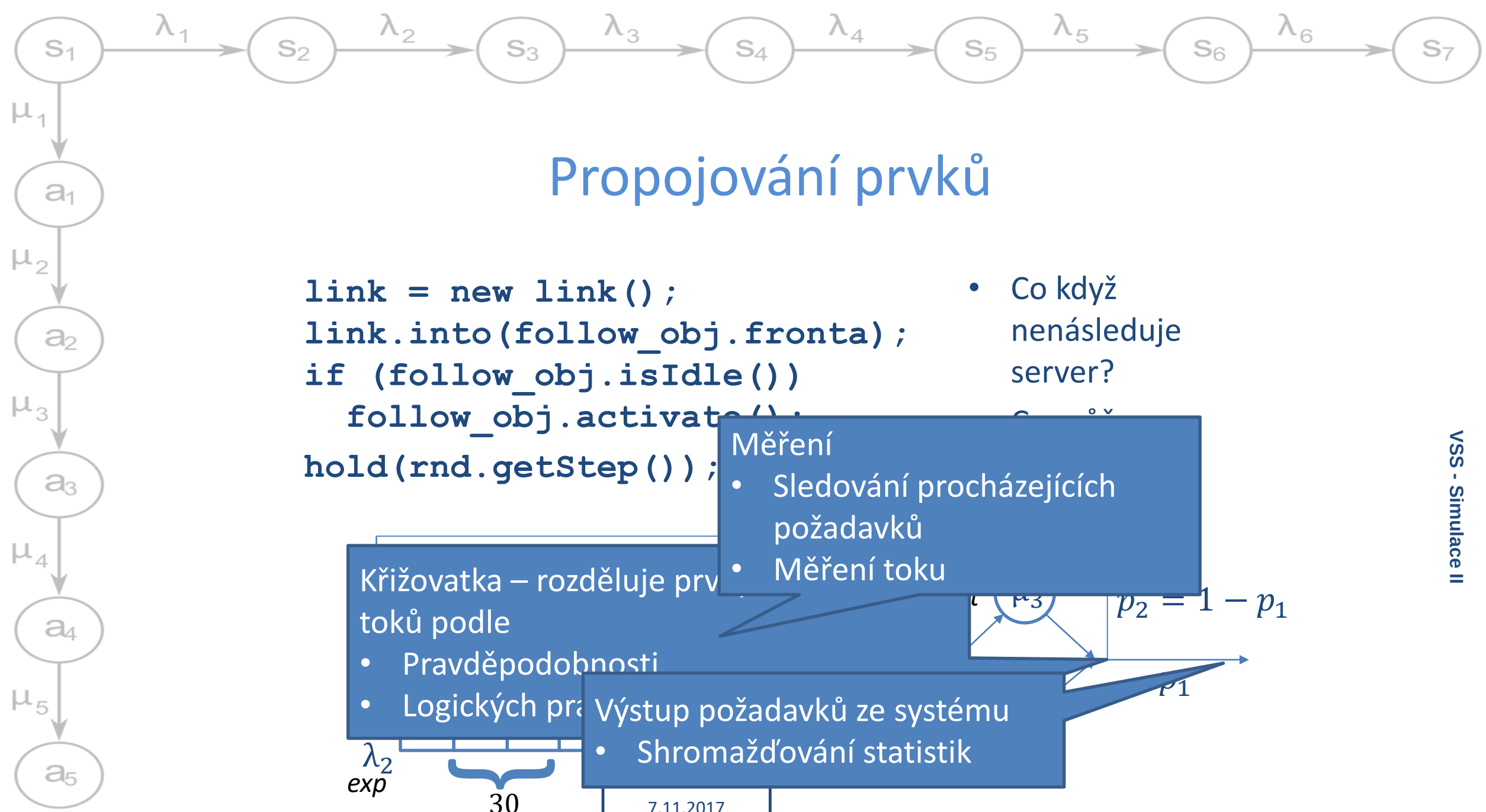
Modelování SHO

- Matematické modely pro M/M/1, G/G/1 …
  - Pro složitější systémy jen simulace
  - Lze modelovat libovolné chování příchodu požadavků i uzlů
  - Lze libovolně spojovat uzly (elementární SHO do sítí)



Návrh simulačního programu

- U knihoven JSim a CSim příklady na jejich použití
  - Ne moc dobé příklady na objektový návrh → v semestrálce se snažte navrhovat lépe
- Knihovna neumožňuje snadné propojování různých typů objektů
  - Základní objekty jsou jen *proces* a *fronta*, nic dalšího
  - Jaké situace mohou nastat?



Propojování prvků

```
link = new link();
link.into(follow_obj.fronta);
if (follow_obj.isIdle())
  follow_obj.activate();
hold(rnd.getStep());
```

- Co když nenásleduje server?

Měření
- Sledování procházejících požadavků
- Měření toku

Křižovatka – rozděluje prv
toků podle
- Pravděpodobnosti
- Logických pra

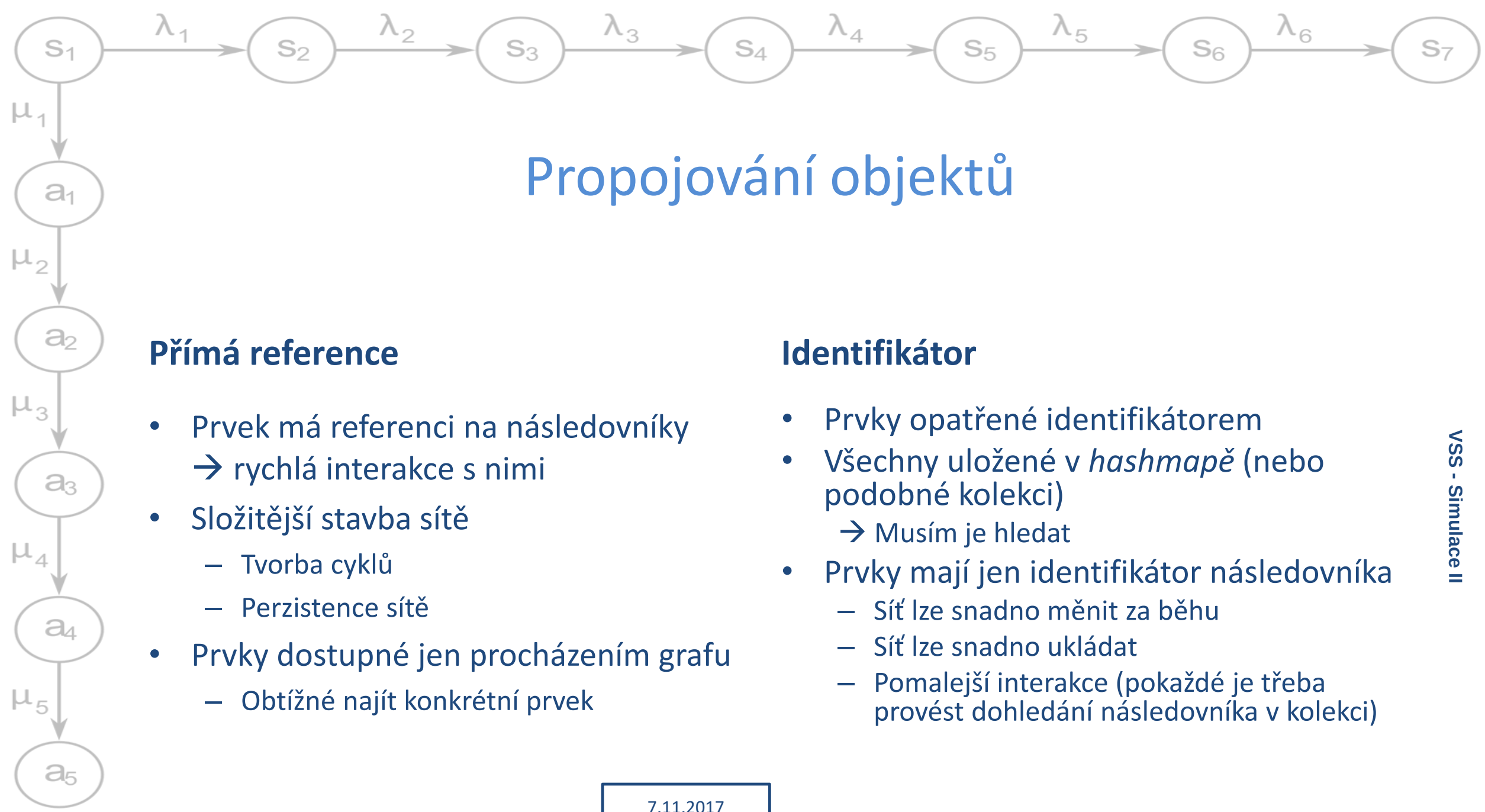
Výstup požadavků ze systému
- Shromažďování statistik

λ_2 *exp*

30

$p_2 = 1 - p_1$

p_1



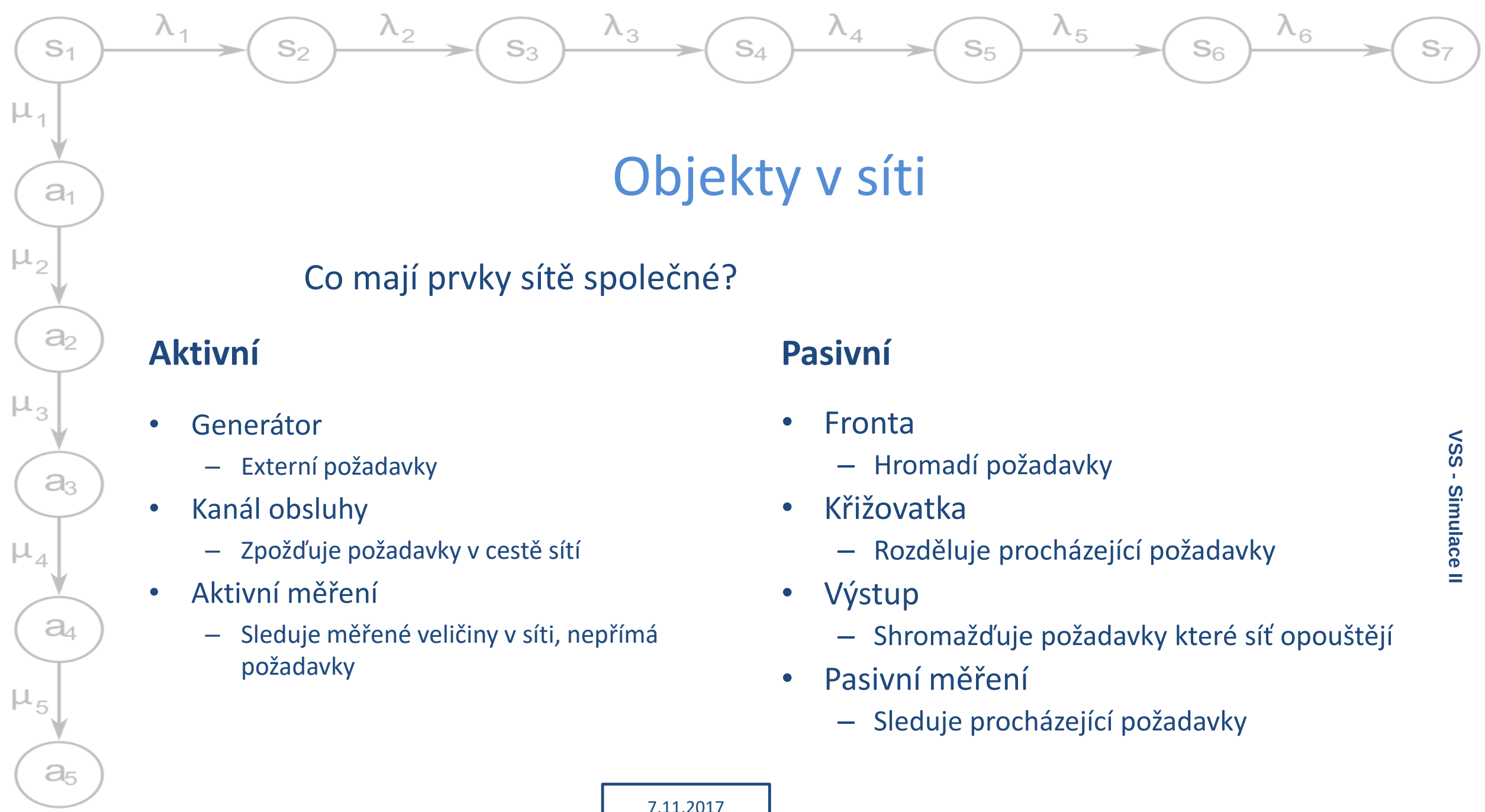
Propojování objektů

Přímá reference

- Prvek má referenci na následovníky
  → rychlá interakce s nimi
- Složitější stavba sítě
  - Tvorba cyklů
  - Perzistence sítě
- Prvky dostupné jen procházením grafu
  - Obtížné najít konkrétní prvek

Identifikátor

- Prvky opatřené identifikátorem
- Všechny uložené v *hashmapě* (nebo podobné kolekci)
  → Musím je hledat
- Prvky mají jen identifikátor následovníka
  - Síť lze snadno měnit za běhu
  - Síť lze snadno ukládat
  - Pomalejší interakce (pokaždé je třeba provést dohledání následovníka v kolekci)



Objekty v síti

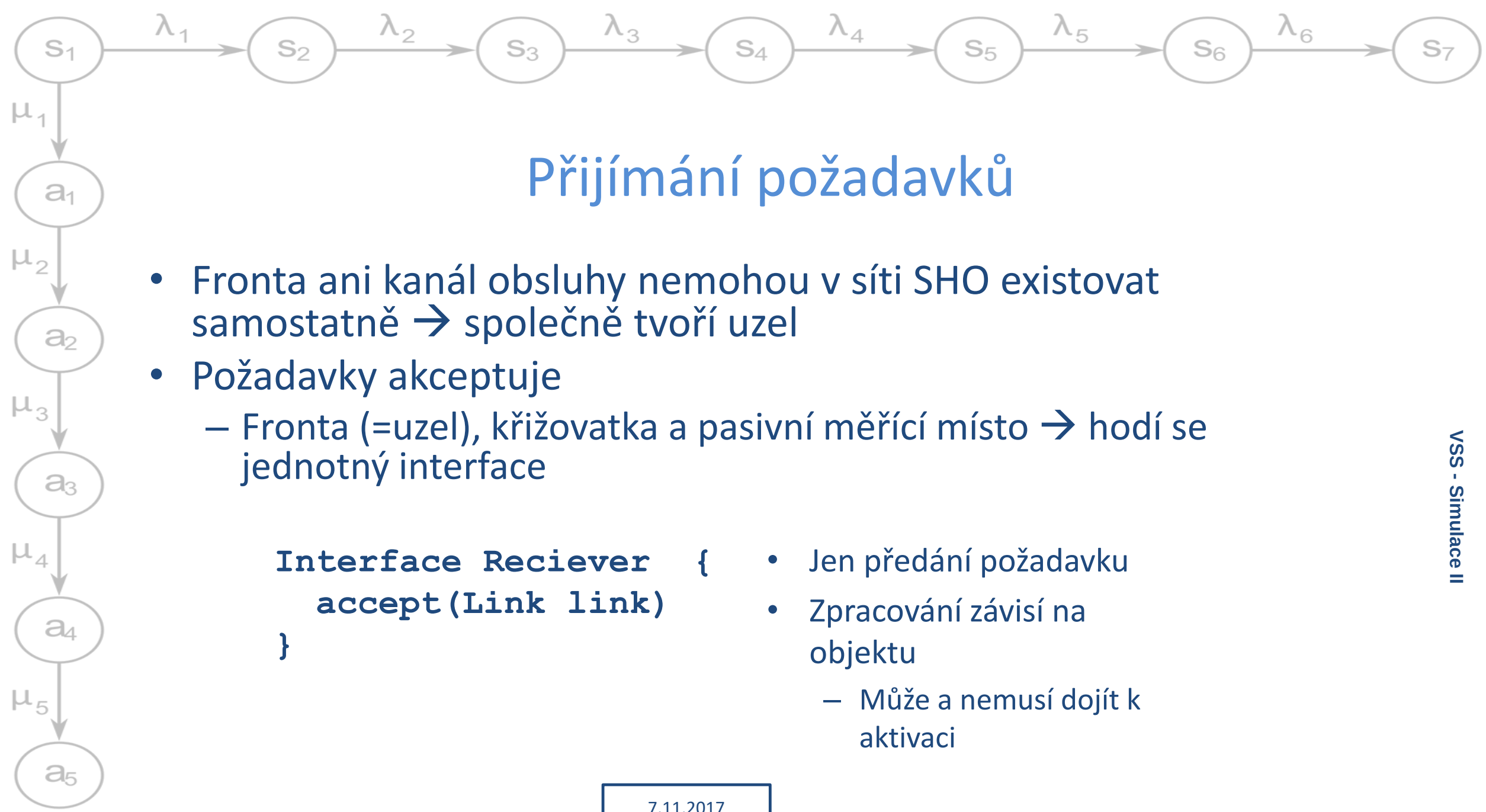
Co mají prvky sítě společné?

Aktivní

- Generátor
  - Externí požadavky
- Kanál obsluhy
  - Zpožďuje požadavky v cestě sítí
- Aktivní měření
  - Sleduje měřené veličiny v síti, nepřímá požadavky

Pasivní

- Fronta
  - Hromadí požadavky
- Křižovatka
  - Rozděluje procházející požadavky
- Výstup
  - Shromažďuje požadavky které síť opouštějí
- Pasivní měření
  - Sleduje procházející požadavky



Přijímání požadavků

- Fronta ani kanál obsluhy nemohou v síti SHO existovat samostatně → společně tvoří uzel
- Požadavky akceptuje
  - Fronta (=uzel), křižovatka a pasivní měřící místo → hodí se jednotný interface

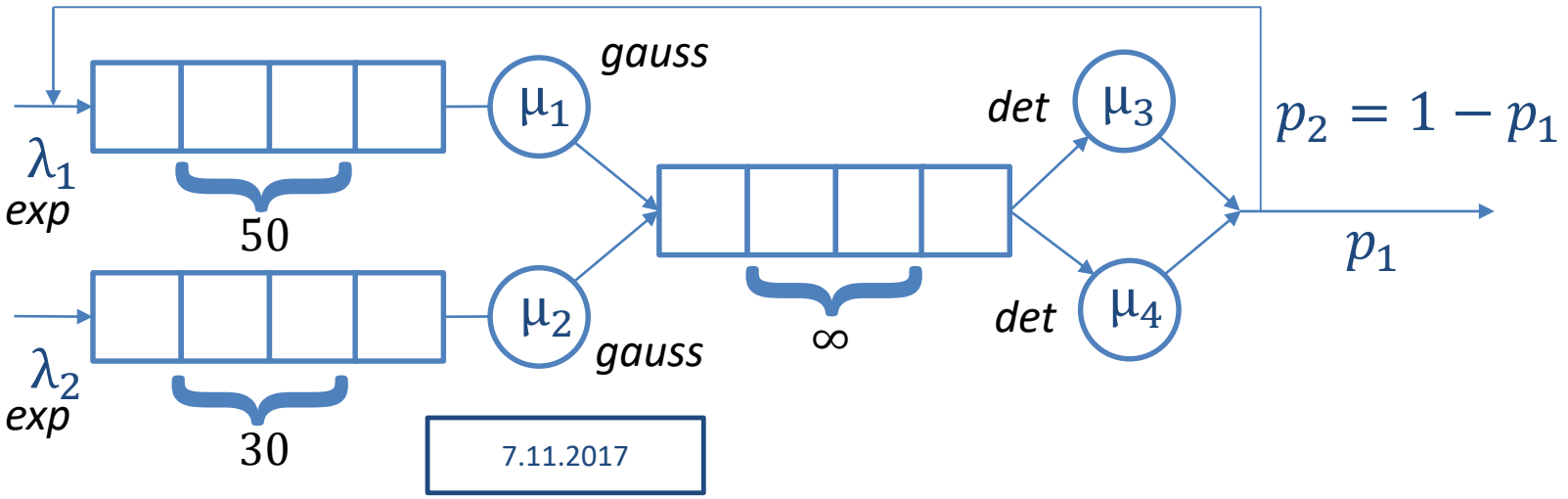
```
Interface Reciever  {
  accept(Link link)
}
```

- Jen předání požadavku
- Zpracování závisí na objektu
  - Může a nemusí dojít k aktivaci

Propojení

- Každý prvek potřebuje odkaz na následníka (atribut / pole **`dalšíPřijímač`**)
  - Kromě výstupu
  - Propojení referencí / identifikátorem
  - Možný kandidát na další rozhraní (**`setNextReceiver()`**)

→ lze sestavit libovolnou síť



Základní struktura prvků

Generátor

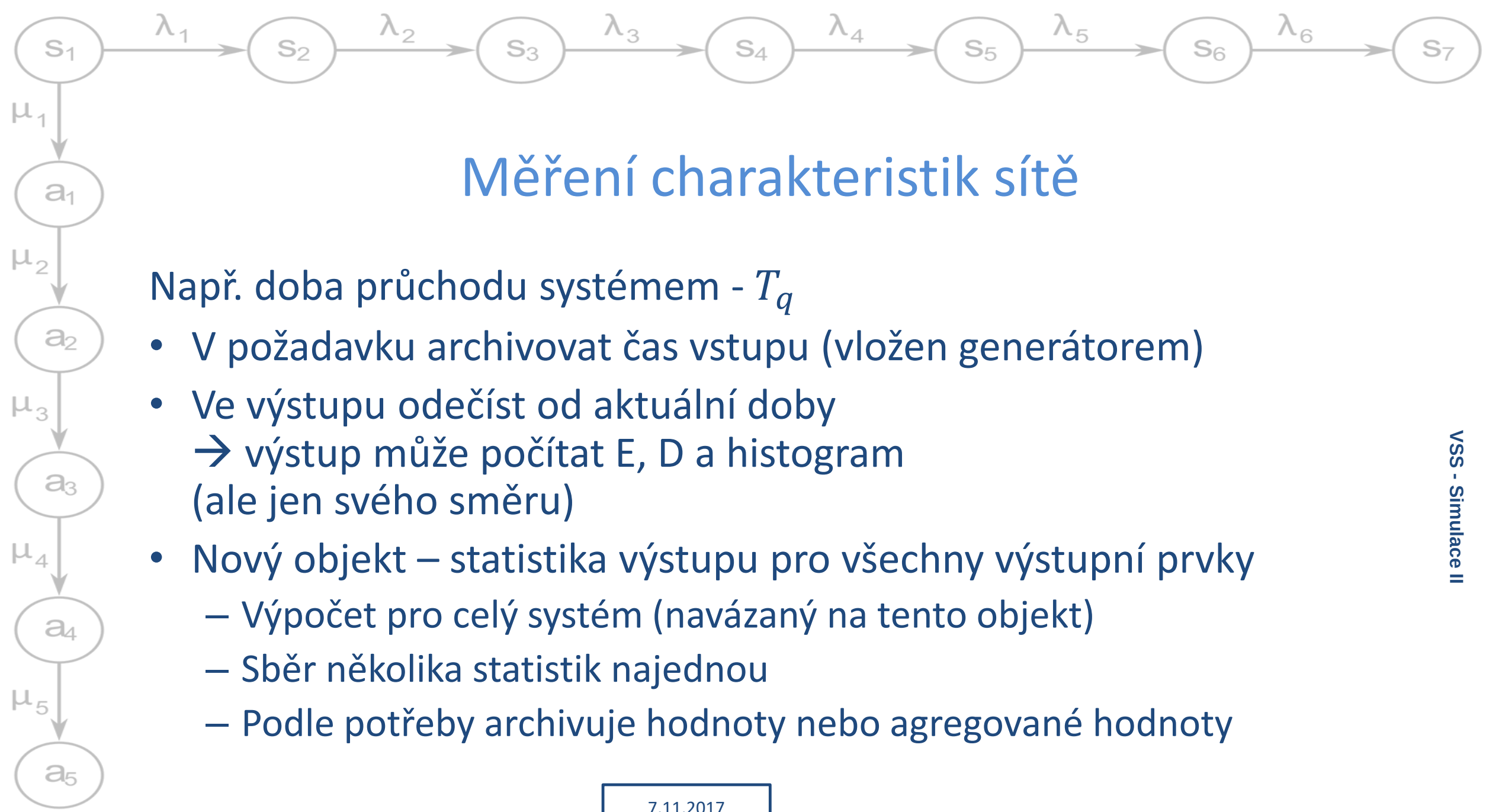
```
link = new Link();
nextRec.accept(link);
hold(rnd.getStep());
```

Uzel (fronta + server)

```
accept(Link link) {
  link.into(myQueue);
  if (this.isIdle())
    activate();
}
```

Křižovatka

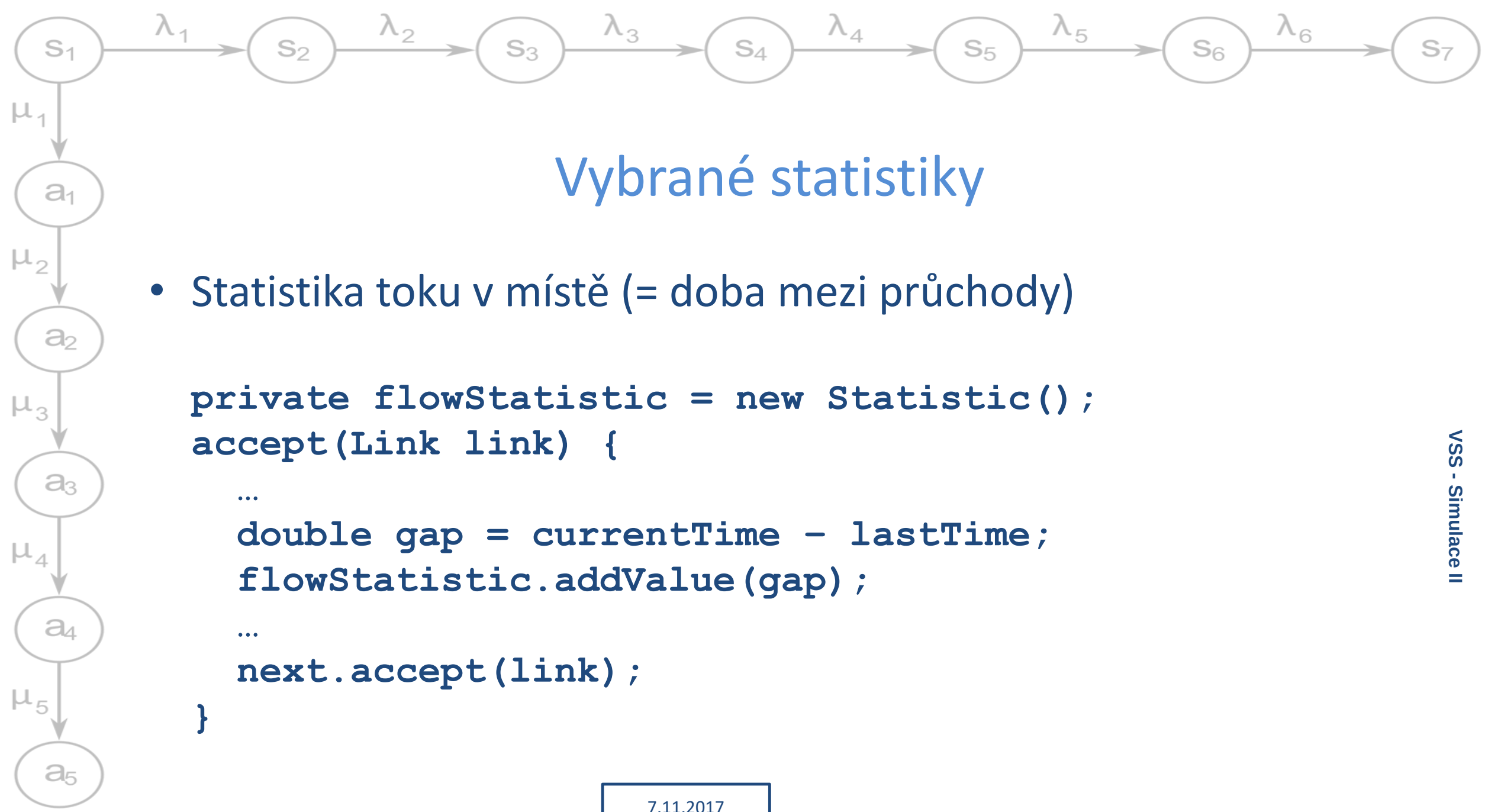
```
accept(Link link) {
  Reciever follower = chooseFollower();
  follower.accept(link);
}
```



Měření charakteristik sítě

Např. doba průchodu systémem - T_q

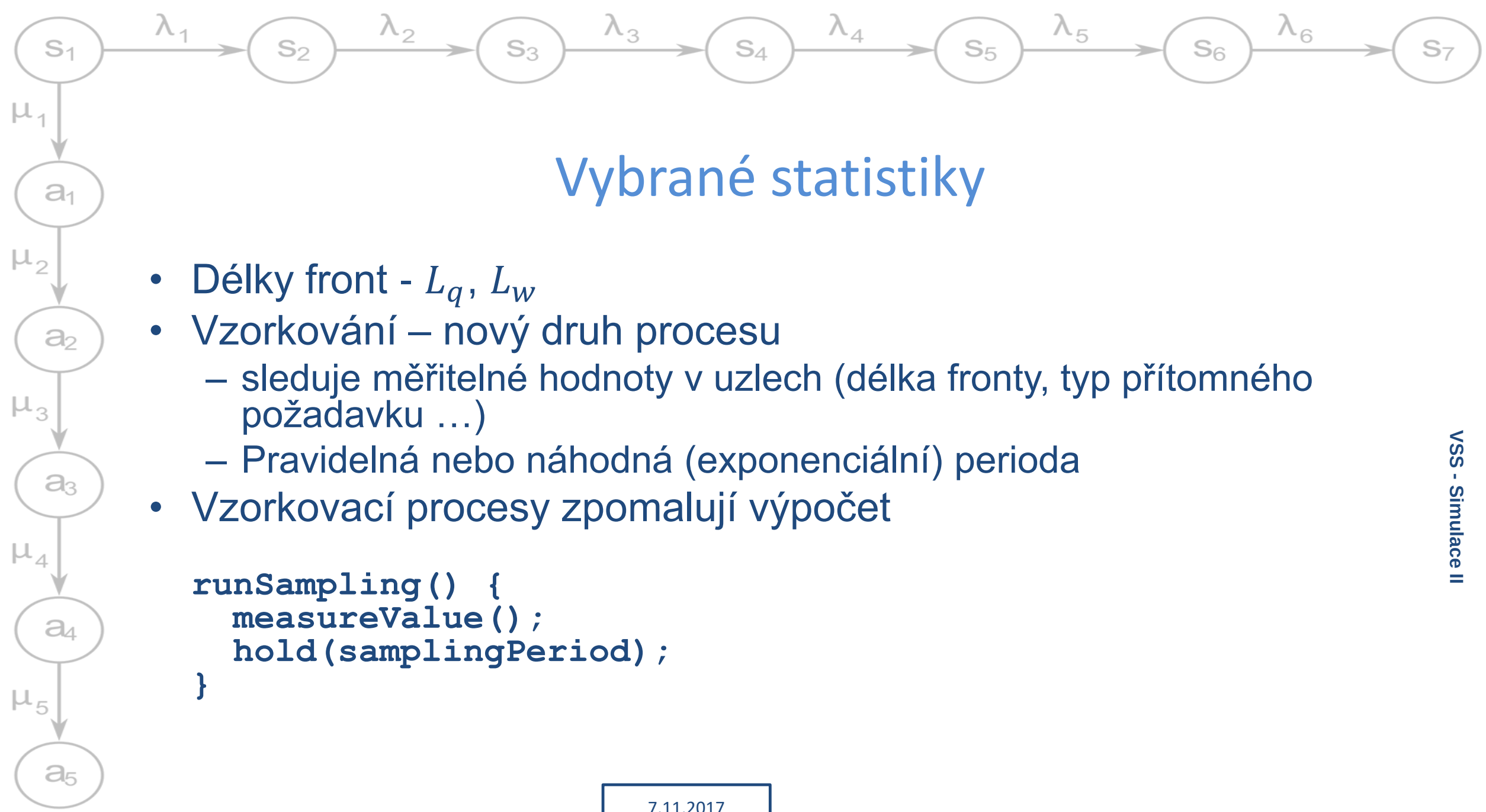
- V požadavku archivovat čas vstupu (vložen generátorem)
- Ve výstupu odečíst od aktuální doby
  → výstup může počítat E, D a histogram
  (ale jen svého směru)
- Nový objekt – statistika výstupu pro všechny výstupní prvky
  - Výpočet pro celý systém (navázaný na tento objekt)
  - Sběr několika statistik najednou
  - Podle potřeby archivuje hodnoty nebo agregované hodnoty



Vybrané statistiky

- Statistika toku v místě (= doba mezi průchody)

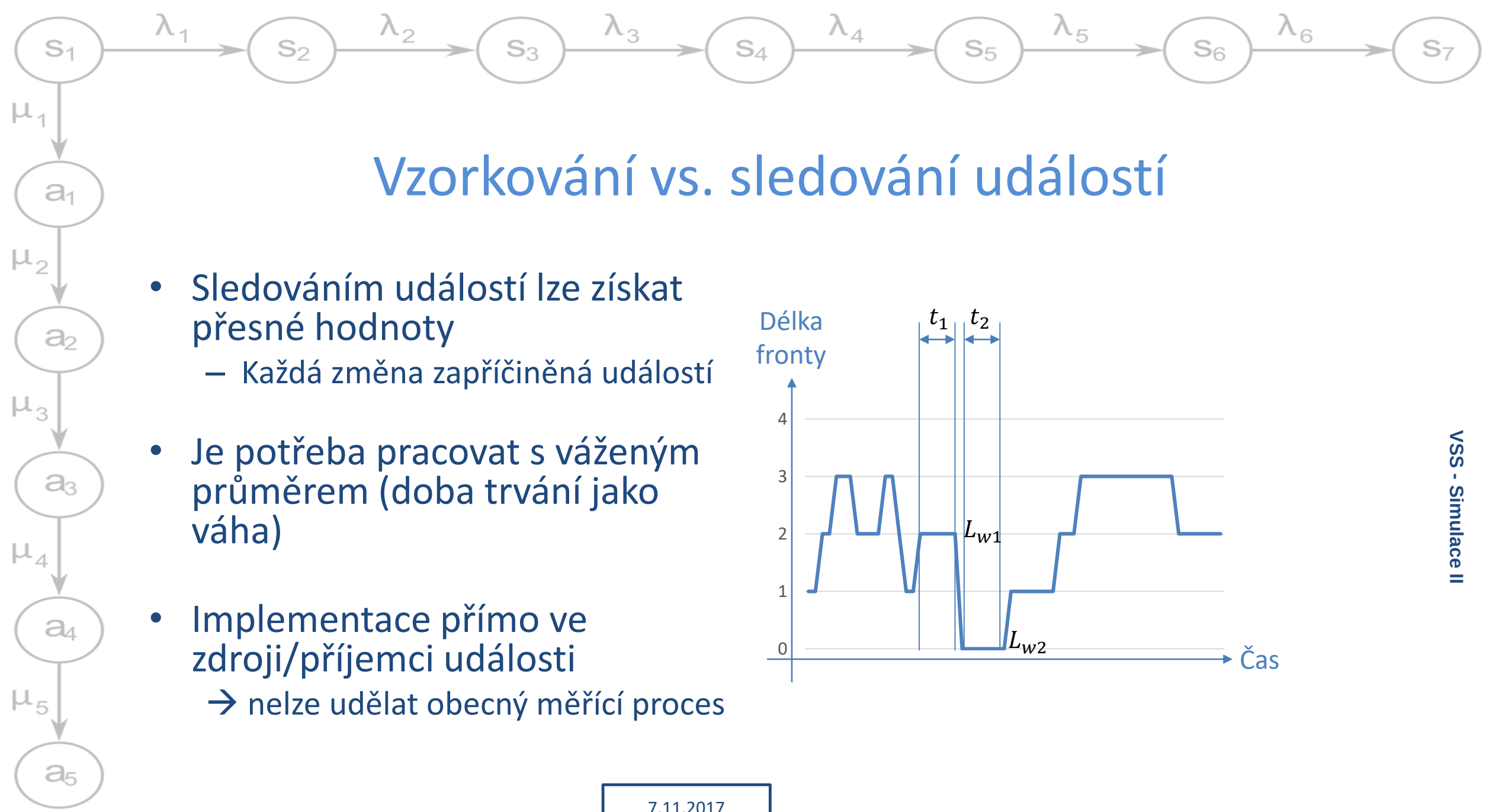
```
private flowStatistic = new Statistic();
accept(Link link) {
   …
   double gap = currentTime – lastTime;
   flowStatistic.addValue(gap);
   …
   next.accept(link);
}
```



Vybrané statistiky

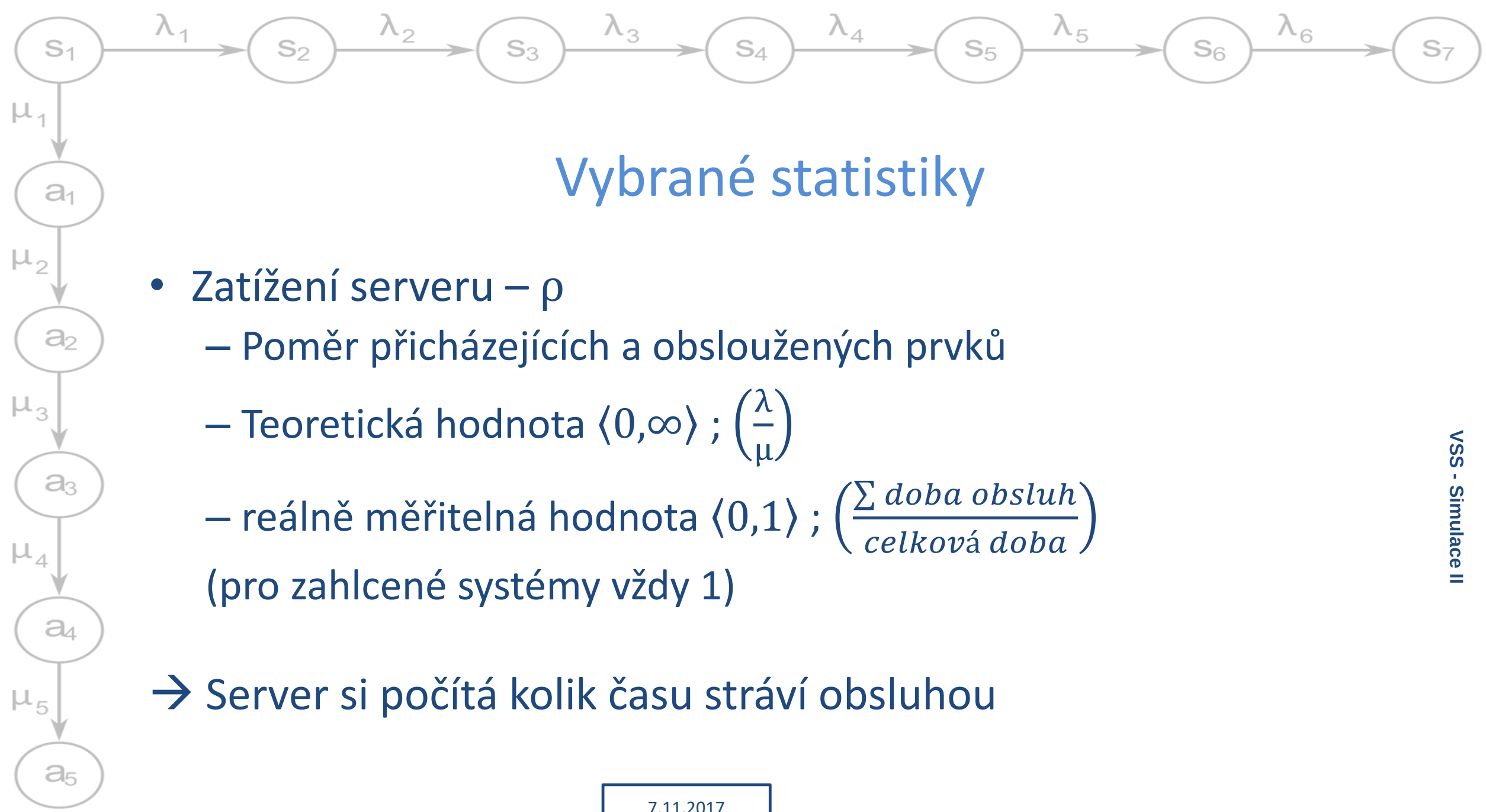
- Délky front - L_q, L_w
- Vzorkování – nový druh procesu
  - sleduje měřitelné hodnoty v uzlech (délka fronty, typ přítomného požadavku …)
  - Pravidelná nebo náhodná (exponenciální) perioda
- Vzorkovací procesy zpomalují výpočet

```
runSampling() {
  measureValue();
  hold(samplingPeriod);
}
```



Vzorkování vs. sledování událostí

- Sledováním událostí lze získat přesné hodnoty
  - Každá změna zapříčiněná událostí
- Je potřeba pracovat s váženým průměrem (doba trvání jako váha)
- Implementace přímo ve zdroji/příjemci události
  → nelze udělat obecný měřící proces

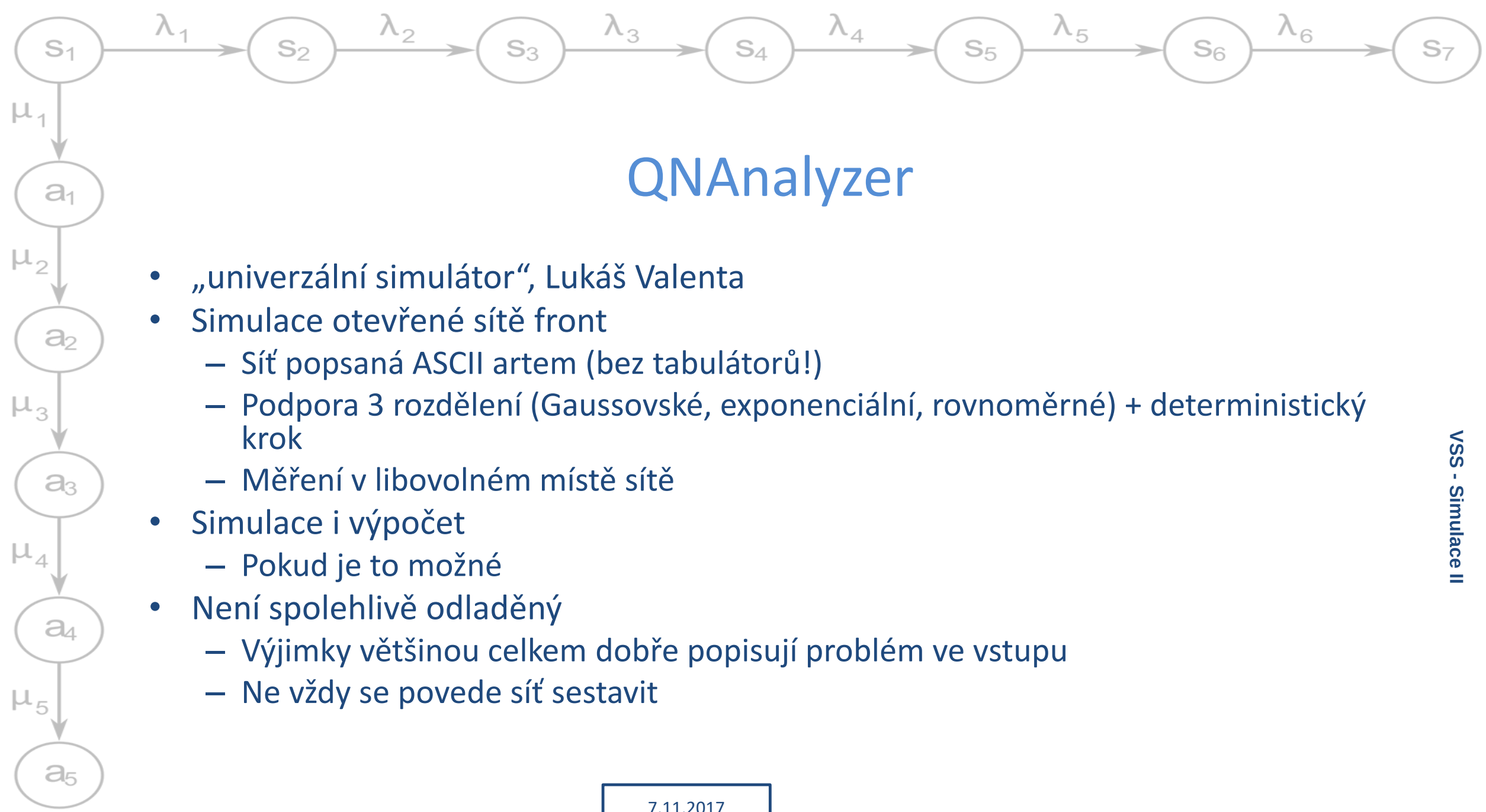


Vybrané statistiky

- Zatížení serveru – ρ
  - Poměr přicházejících a obsloužených prvků
  - Teoretická hodnota $\langle 0,\infty \rangle$; $\left(\frac{\lambda}{\mu}\right)$
  - reálně měřitelná hodnota $\langle 0,1 \rangle$; $\left(\frac{\sum doba\ obsluh}{celková\ doba}\right)$

  (pro zahlcené systémy vždy 1)

→ Server si počítá kolik času stráví obsluhou



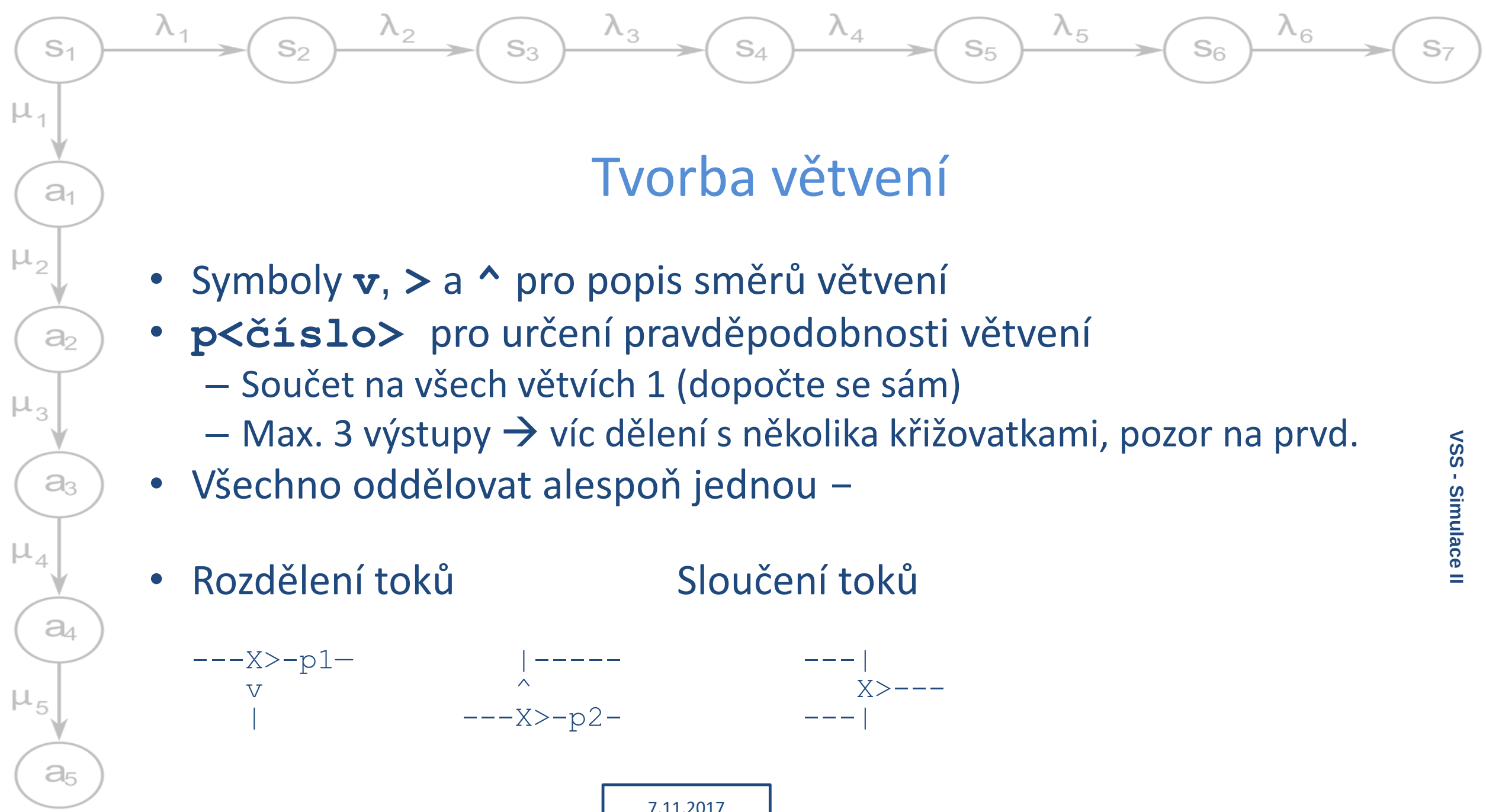
QNAnalyzer

- „univerzální simulátor“, Lukáš Valenta
- Simulace otevřené sítě front
  - Síť popsaná ASCII artem (bez tabulátorů!)
  - Podpora 3 rozdělení (Gaussovské, exponenciální, rovnoměrné) + deterministický krok
  - Měření v libovolném místě sítě
- Simulace i výpočet
  - Pokud je to možné
- Není spolehlivě odladěný
  - Výjimky většinou celkem dobře popisují problém ve vstupu
  - Ne vždy se povede síť sestavit

Objekty v síti

- **S<číslo>** - vstup požadavků
- **O<číslo>** - kanál obsluhy
- **E<číslo>** - terminátor
- **M<číslo>** - měřící místo
- **X** – křižovatka (max. 4 spojené proudy)

- Čísla k identifikaci elementů
  - Pro výstup statistik
  - Pro nastavení parametrů (pod sítí)



Tvorba větvení

- Symboly **v**, **>** a **^** pro popis směrů větvení
- **p<číslo>** pro určení pravděpodobnosti větvení
  - Součet na všech větvích 1 (dopočte se sám)
  - Max. 3 výstupy → víc dělení s několika křižovatkami, pozor na prvd.
- Všechno oddělovat alespoň jednou –

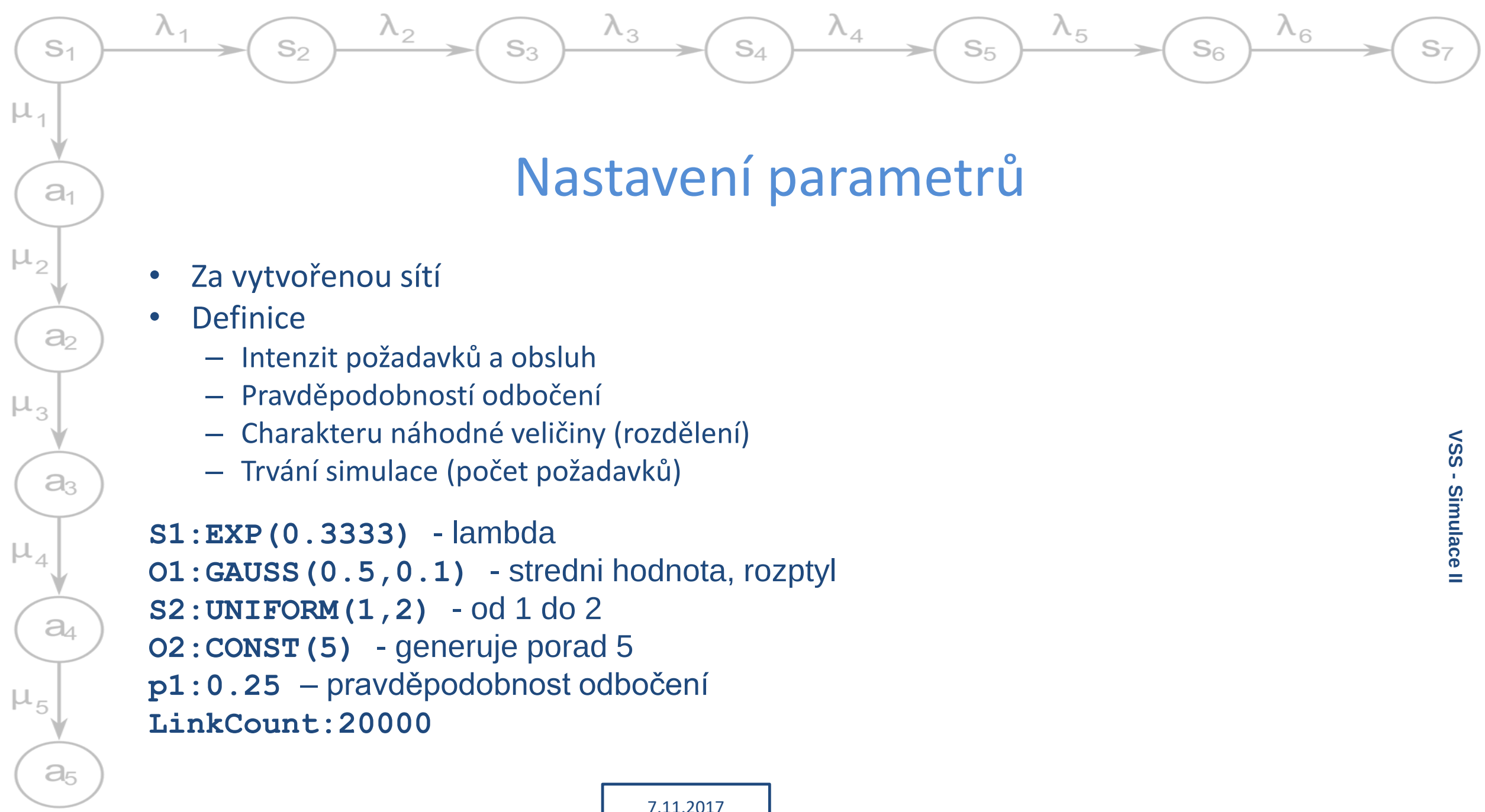
- Rozdělení toků

```
---X>-p1-
   v
   |
```

```
   |-----
   ^
---X>-p2-
```

Sloučení toků

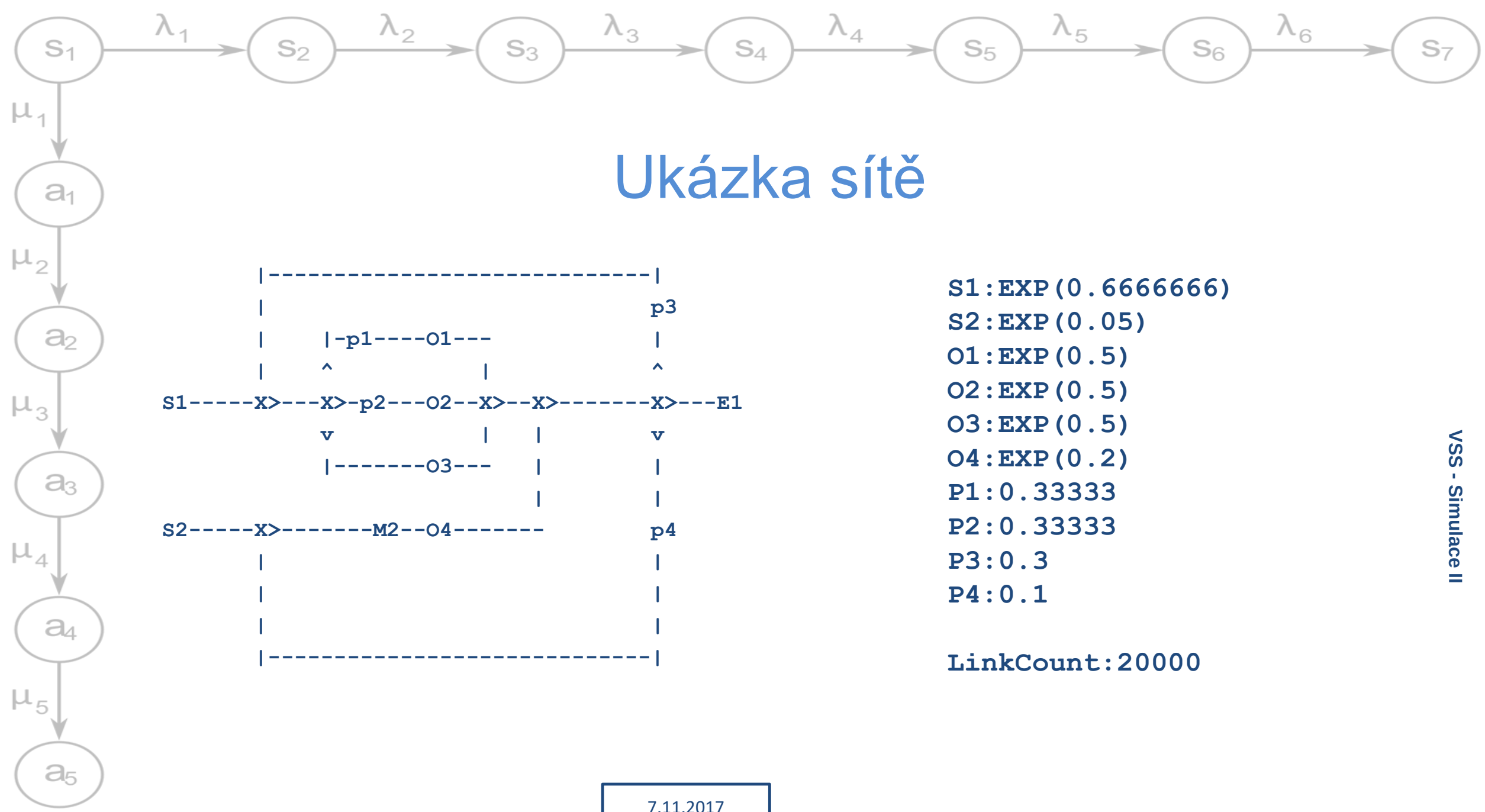
```
---|
   X>---
---|
```



Nastavení parametrů

- Za vytvořenou sítí
- Definice
  - Intenzit požadavků a obsluh
  - Pravděpodobností odbočení
  - Charakteru náhodné veličiny (rozdělení)
  - Trvání simulace (počet požadavků)

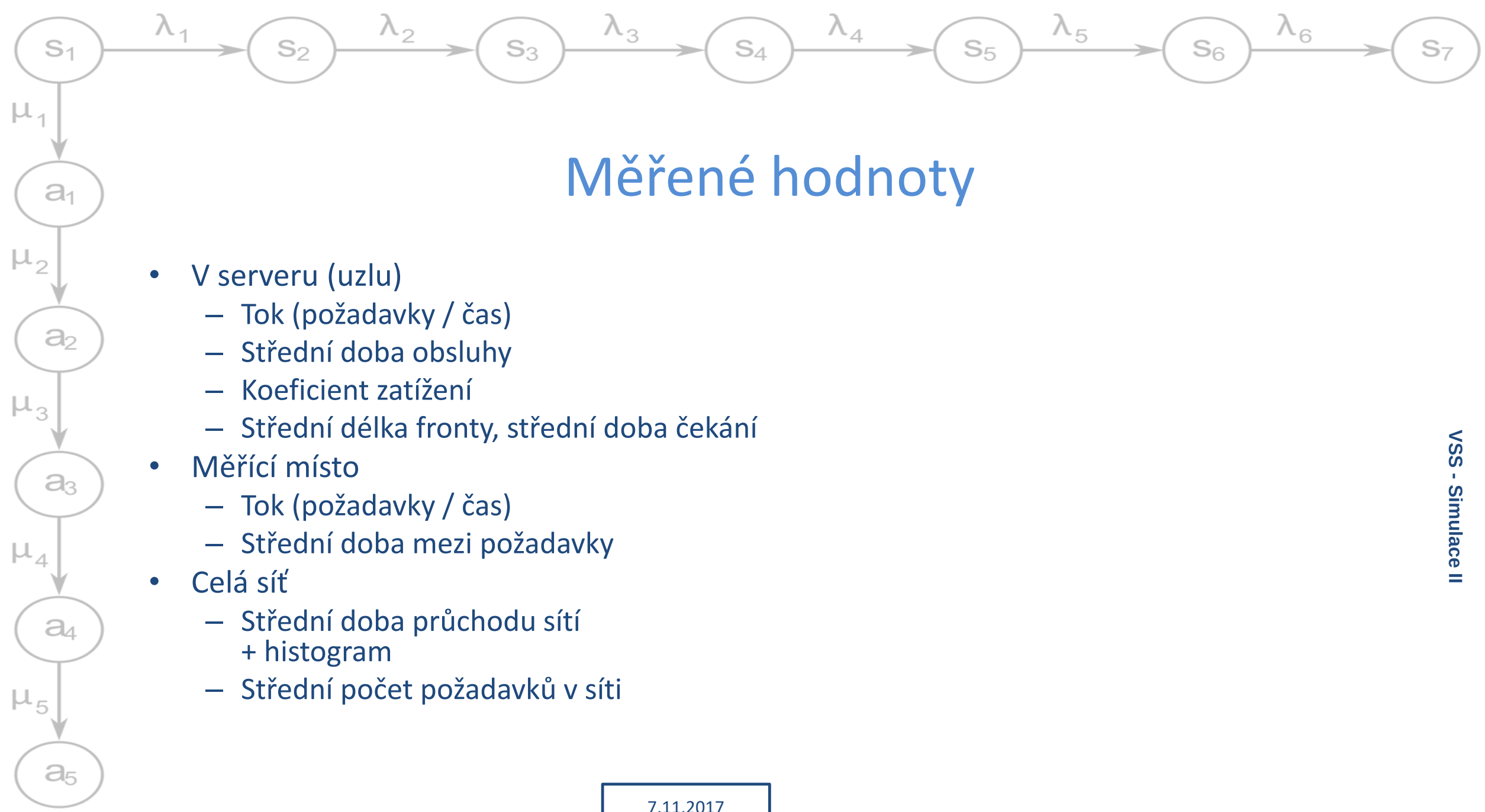
S1:EXP(0.3333) - lambda
O1:GAUSS(0.5,0.1) - stredni hodnota, rozptyl
S2:UNIFORM(1,2) - od 1 do 2
O2:CONST(5) - generuje porad 5
p1:0.25 – pravděpodobnost odbočení
LinkCount:20000



Ukázka sítě

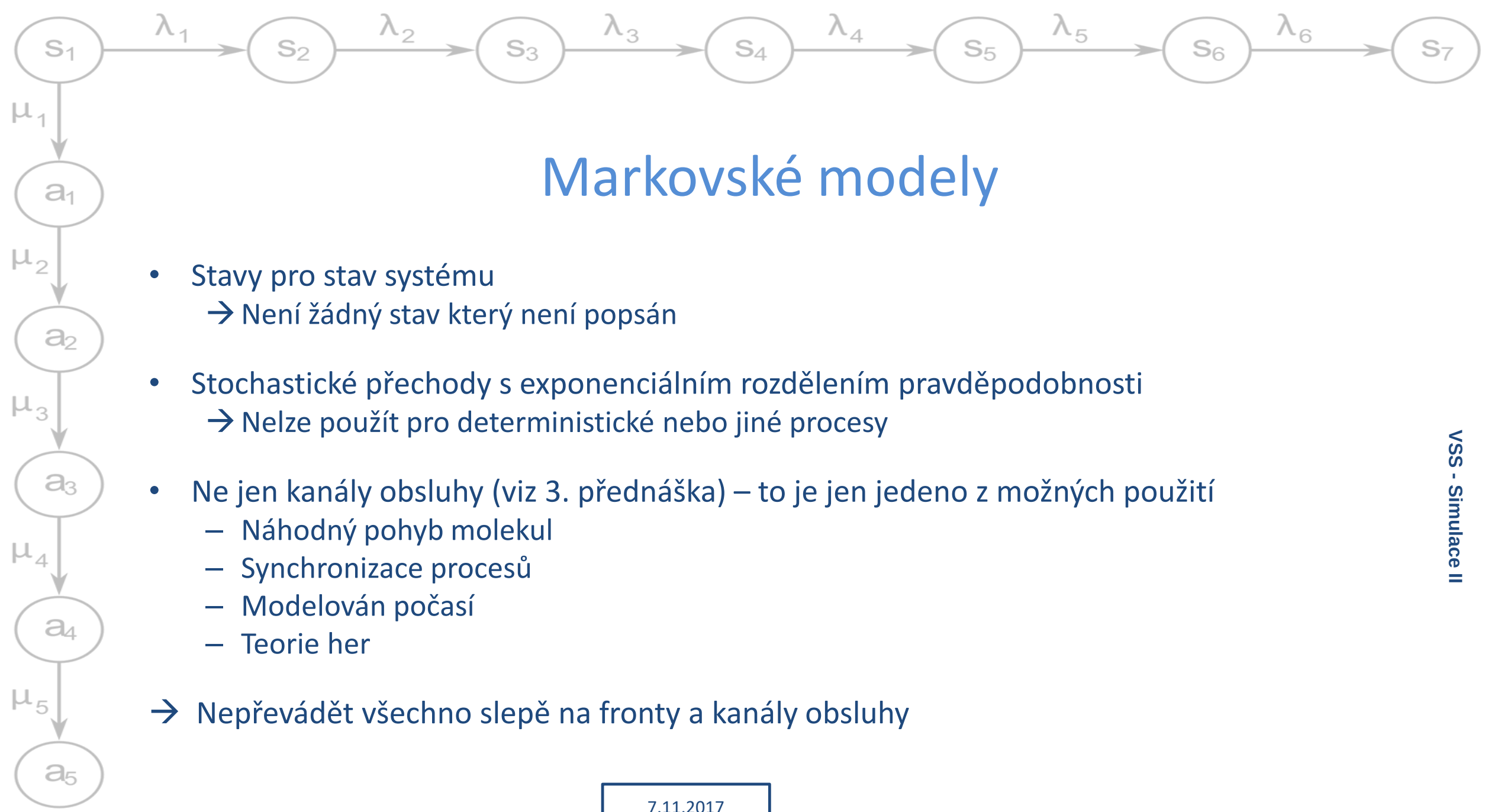
```
        |-----------------------------|
        |                             p3
        |    |-p1----O1---            |
        |    ^           |           ^
S1-----X>---X>-p2---O2--X>--X>-------X>---E1
        |    v           |   |       v
        |    |-------O3---   |       |
        |                    |       |
S2-----X>-------M2--O4-------        p4
        |                            |
        |                            |
        |                            |
        |----------------------------|
```

```
S1:EXP(0.6666666)
S2:EXP(0.05)
O1:EXP(0.5)
O2:EXP(0.5)
O3:EXP(0.5)
O4:EXP(0.2)
P1:0.33333
P2:0.33333
P3:0.3
P4:0.1

LinkCount:20000
```



Měřené hodnoty

- V serveru (uzlu)
  - Tok (požadavky / čas)
  - Střední doba obsluhy
  - Koeficient zatížení
  - Střední délka fronty, střední doba čekání
- Měřící místo
  - Tok (požadavky / čas)
  - Střední doba mezi požadavky
- Celá síť
  - Střední doba průchodu sítí
    + histogram
  - Střední počet požadavků v síti

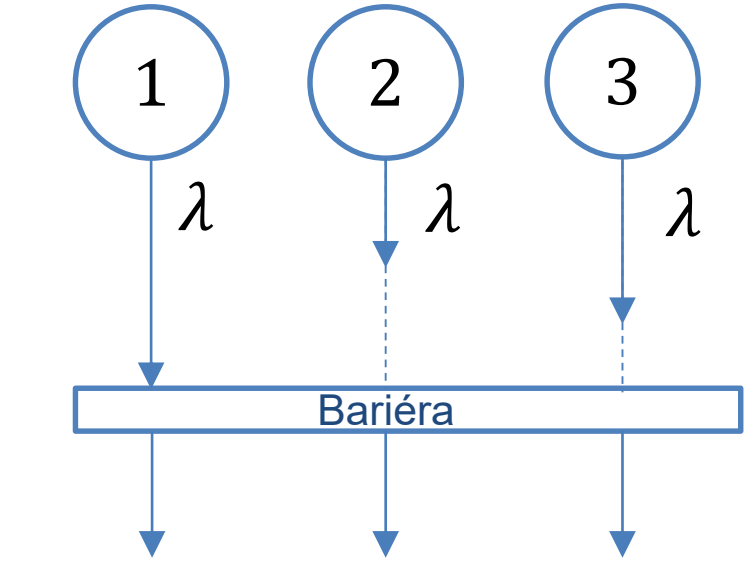


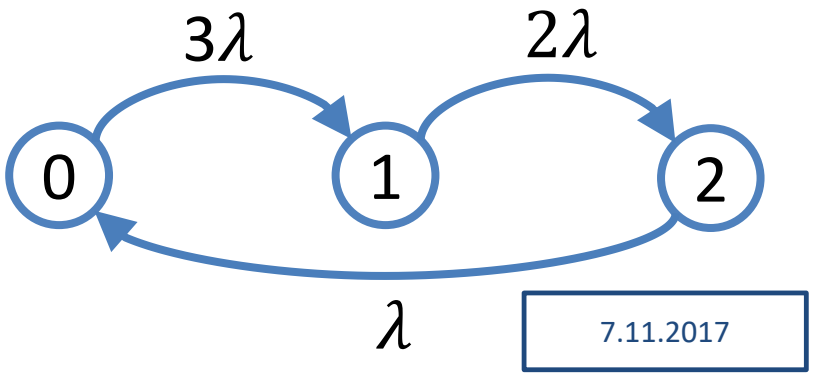
Markovské modely

- Stavy pro stav systému
  → Není žádný stav který není popsán
- Stochastické přechody s exponenciálním rozdělením pravděpodobnosti
  → Nelze použít pro deterministické nebo jiné procesy
- Ne jen kanály obsluhy (viz 3. přednáška) – to je jen jedeno z možných použití
  - Náhodný pohyb molekul
  - Synchronizace procesů
  - Modelován počasí
  - Teorie her

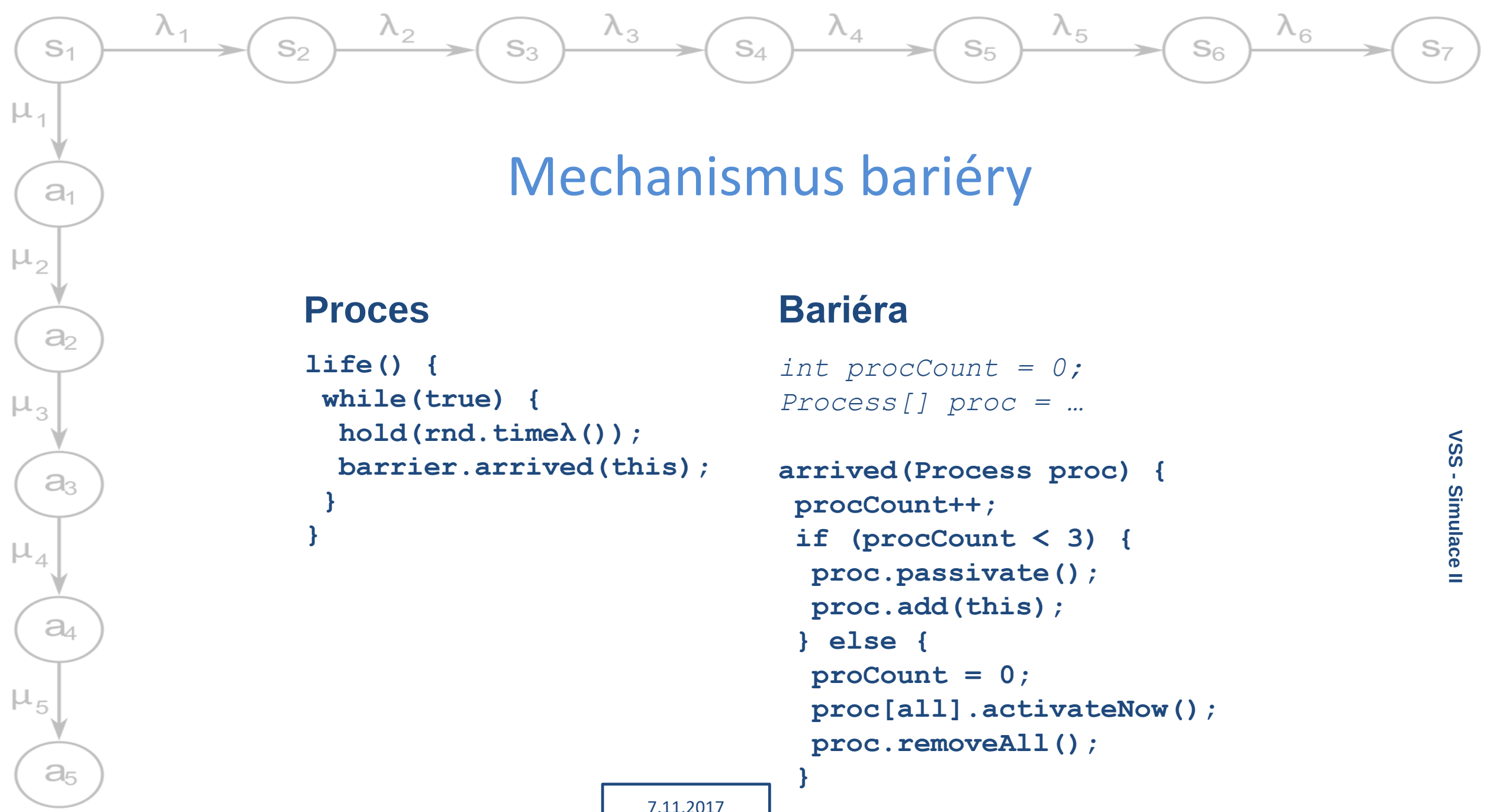
→ Nepřevádět všechno slepě na fronty a kanály obsluhy

Mechanismus bariéry





- Bariéra nezabírá čas → poslední proces který dorazí umožní všem pokračovat ve výpočtu
- „Za bariérou“ se spouští znovu všechna vlákna



Mechanismus bariéry

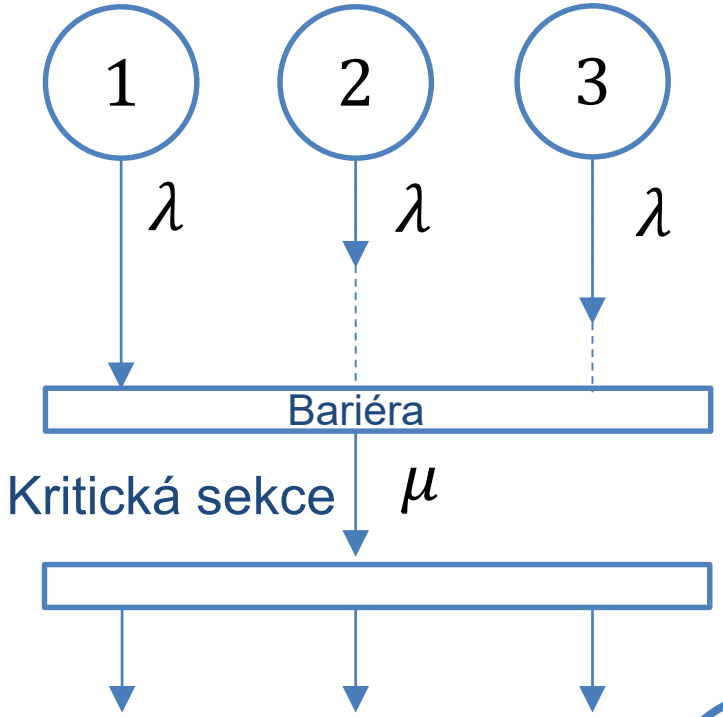
Proces

```
life() {
 while(true) {
  hold(rnd.timeλ());
  barrier.arrived(this);
 }
}
```

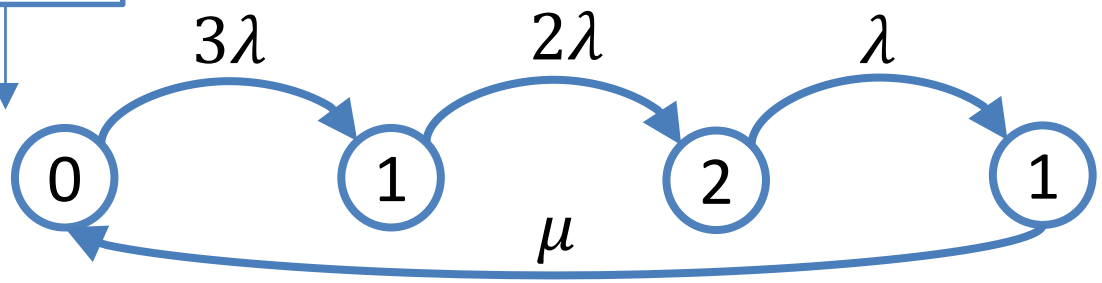
Bariéra

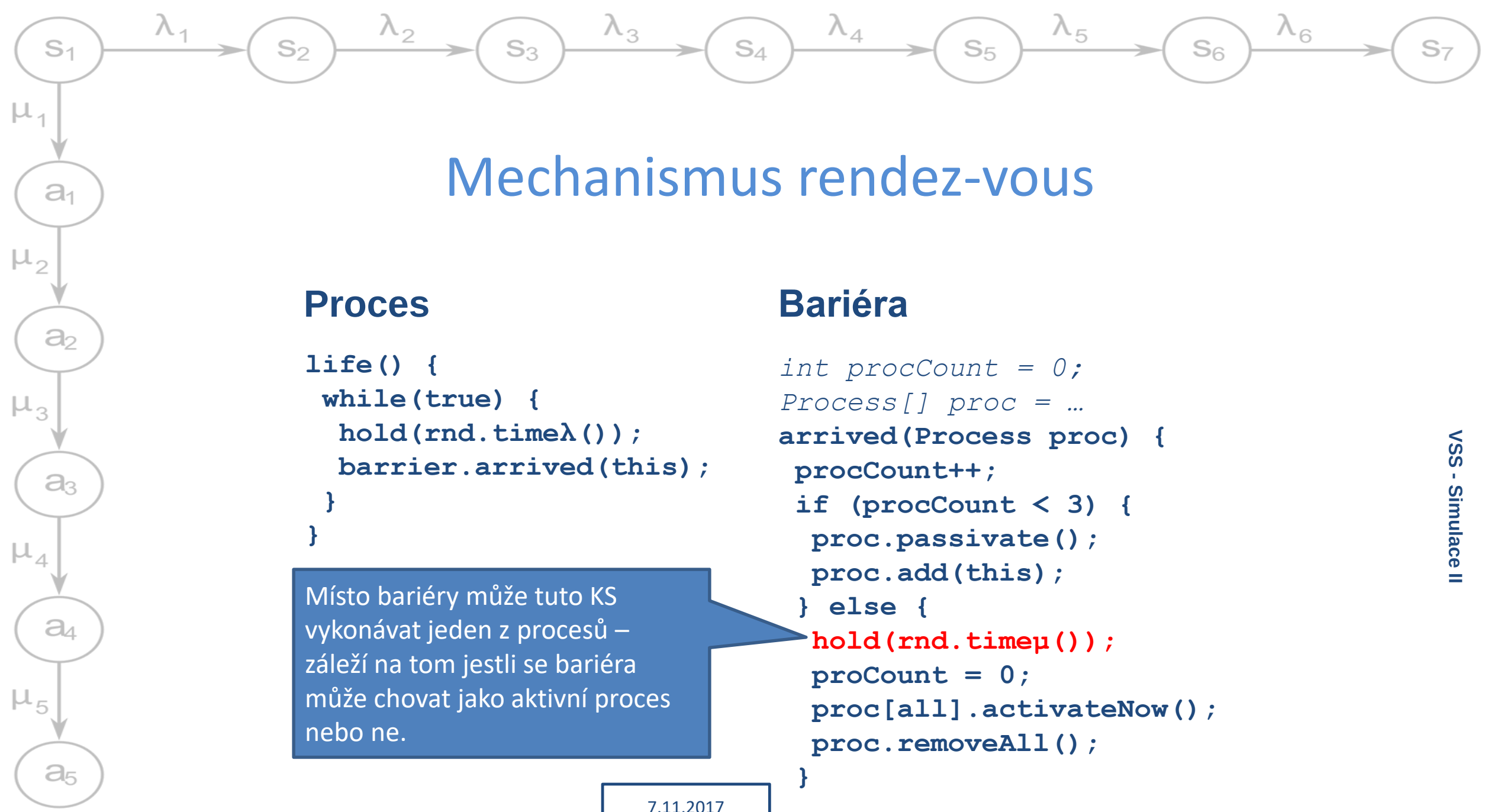
```
int procCount = 0;
Process[] proc = …

arrived(Process proc) {
 procCount++;
 if (procCount < 3) {
  proc.passivate();
  proc.add(this);
 } else {
  proCount = 0;
  proc[all].activateNow();
  proc.removeAll();
 }
```

Bariéra s kritickou sekcí (rendez-vous)



- Bariéra nezabírá čas, ale společná KS ano
- „Za bariérou“ se spouští znovu všechna vlákna





Mechanismus rendez-vous

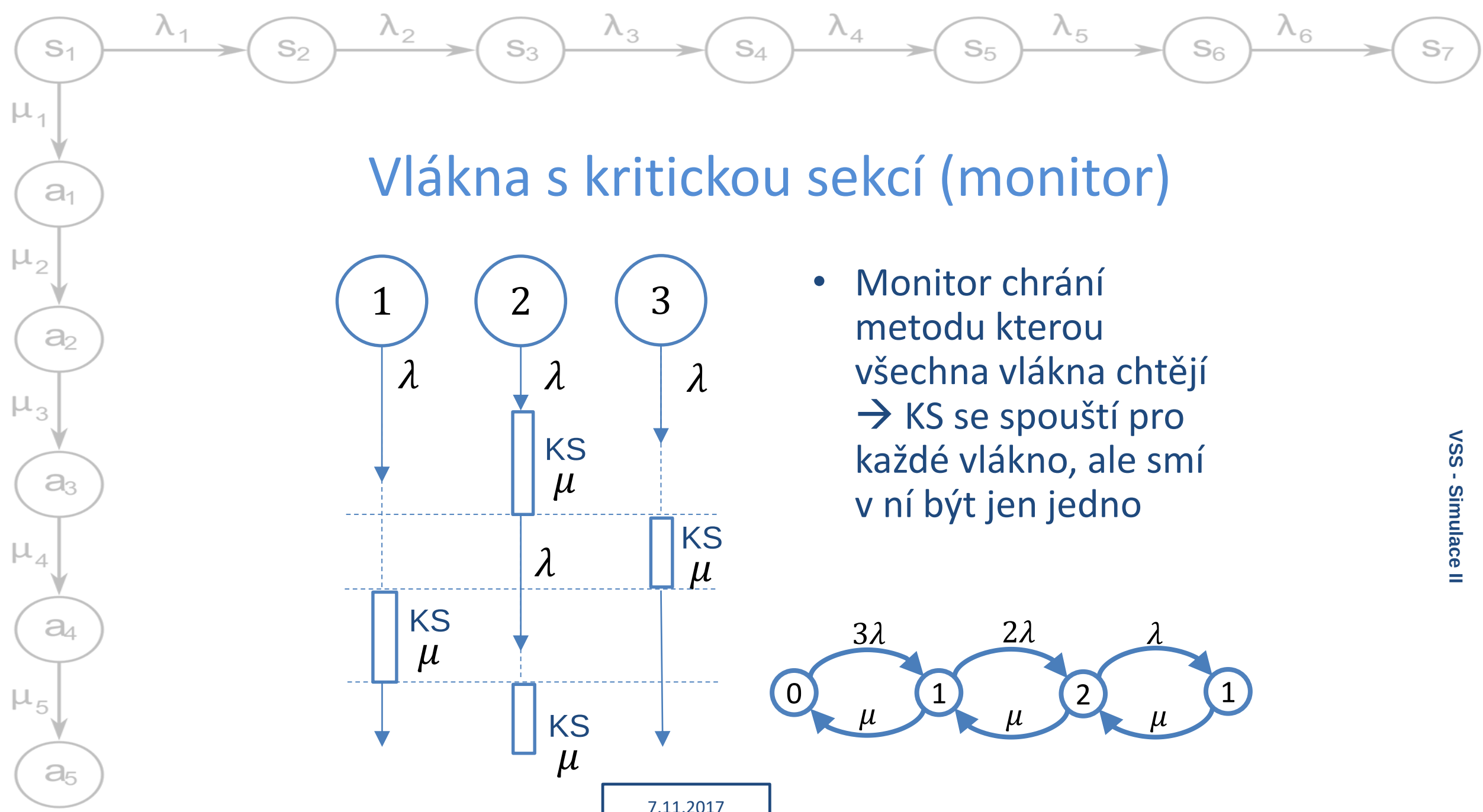
Proces

```
life() {
 while(true) {
  hold(rnd.timeλ());
  barrier.arrived(this);
 }
}
```

Místo bariéry může tuto KS vykonávat jeden z procesů – záleží na tom jestli se bariéra může chovat jako aktivní proces nebo ne.

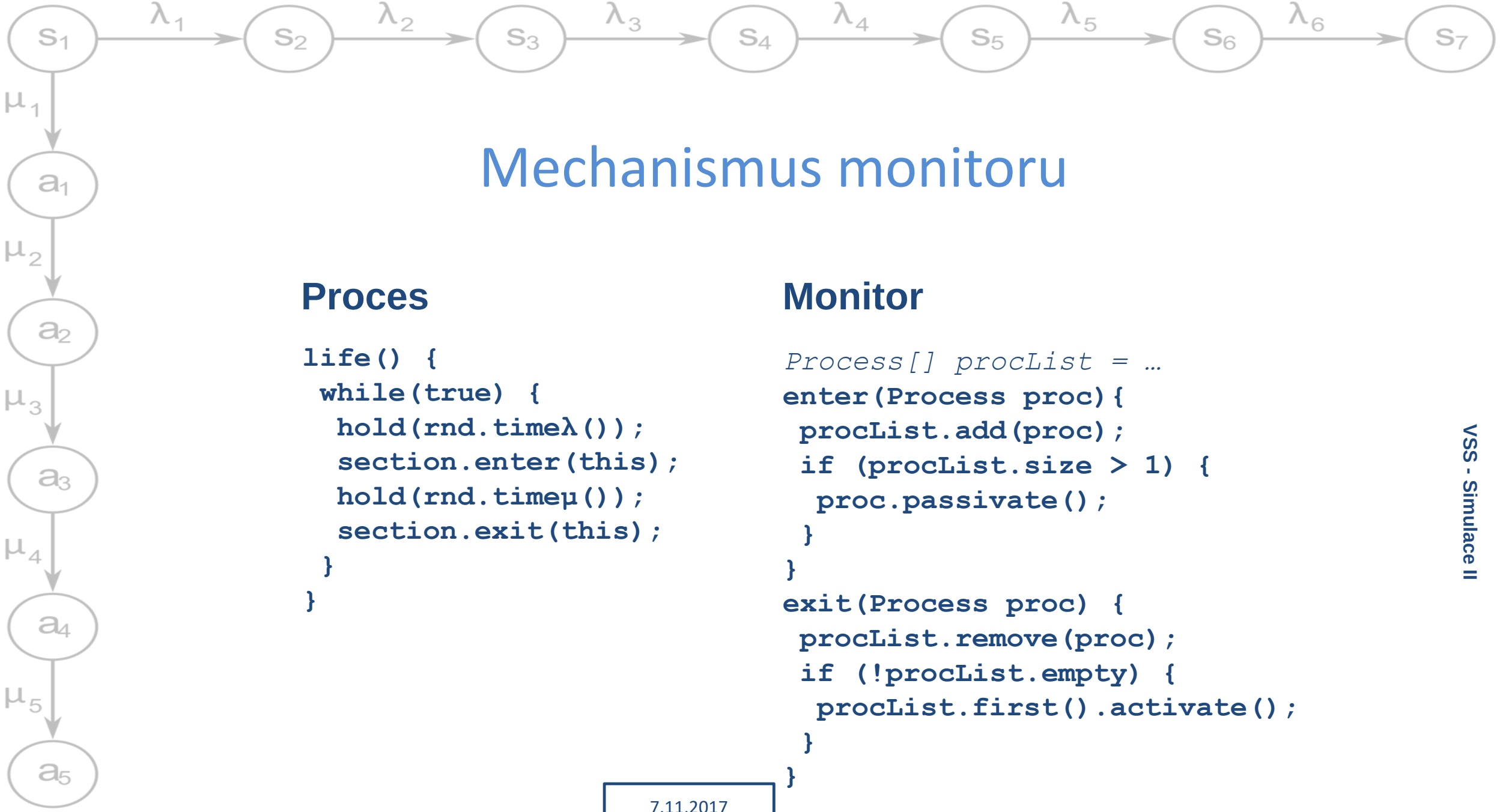
Bariéra

```
int procCount = 0;
Process[] proc = …
arrived(Process proc) {
 procCount++;
 if (procCount < 3) {
  proc.passivate();
  proc.add(this);
 } else {
  hold(rnd.timeμ());
  proCount = 0;
  proc[all].activateNow();
  proc.removeAll();
 }
```



Vlákna s kritickou sekcí (monitor)

- Monitor chrání metodu kterou všechna vlákna chtějí → KS se spouští pro každé vlákno, ale smí v ní být jen jedno



Mechanismus monitoru

Proces

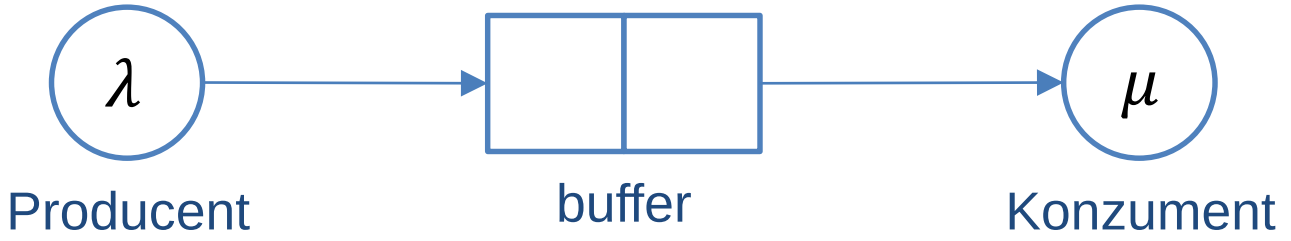
```
life() {
 while(true) {
  hold(rnd.timeλ());
  section.enter(this);
  hold(rnd.timeμ());
  section.exit(this);
 }
}
```

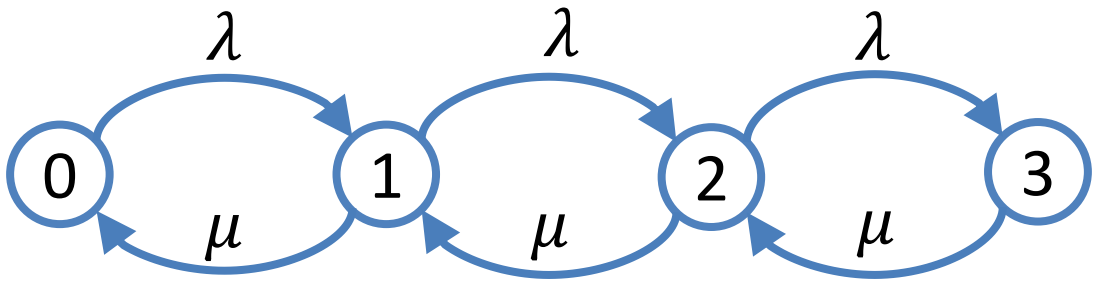
Monitor

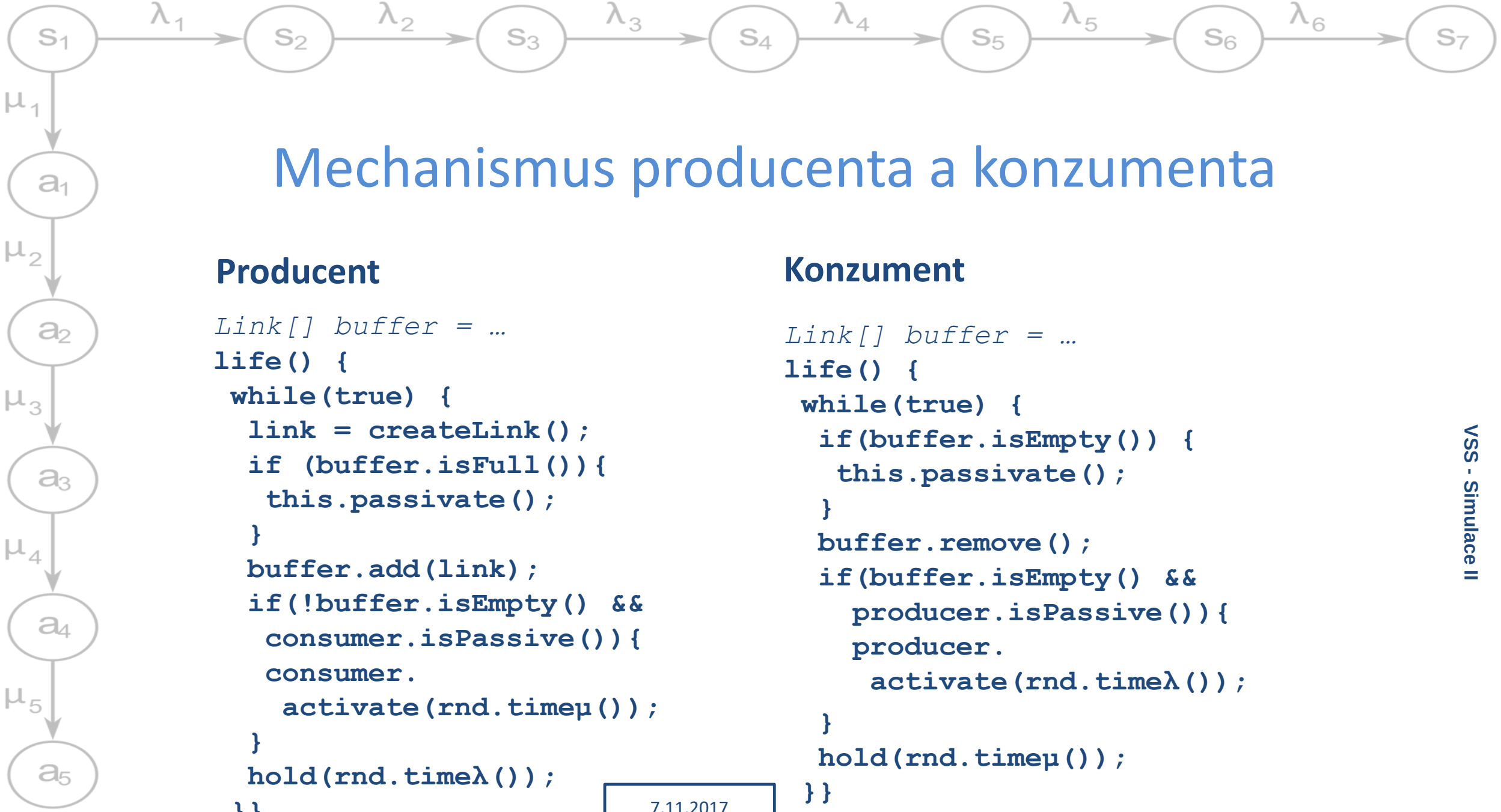
```
Process[] procList = …
enter(Process proc){
 procList.add(proc);
 if (procList.size > 1) {
  proc.passivate();
 }
}
exit(Process proc) {
 procList.remove(proc);
 if (!procList.empty) {
  procList.first().activate();
 }
}
```

Producent – konzument, zasílání zpráv

- Producent nevkládá do plné fronty a nemá další buffer, zastavuje činnost







Mechanismus producenta a konzumenta

Producent

```
Link[] buffer = …
life() {
 while(true) {
  link = createLink();
  if (buffer.isFull()){
   this.passivate();
  }
  buffer.add(link);
  if(!buffer.isEmpty() &&
   consumer.isPassive()){
   consumer.
    activate(rnd.timeμ());
  }
  hold(rnd.timeλ());
 }}
```

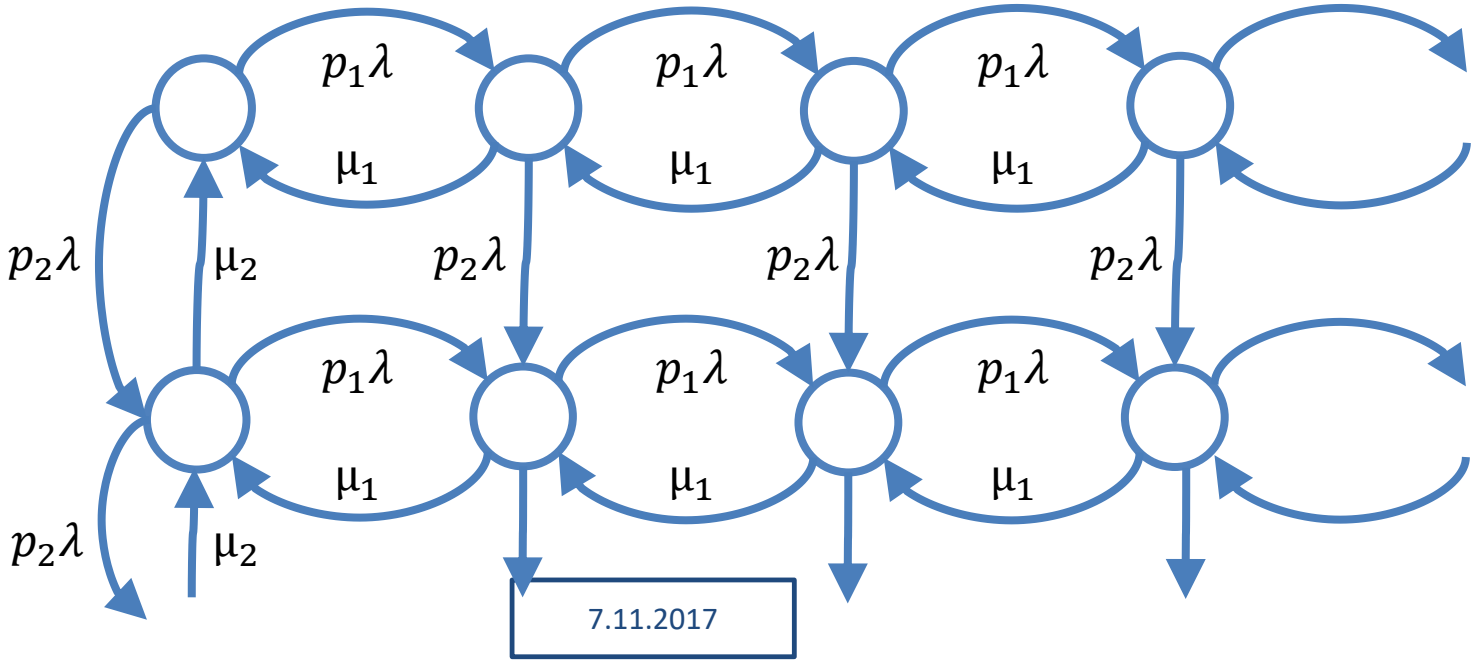
Konzument

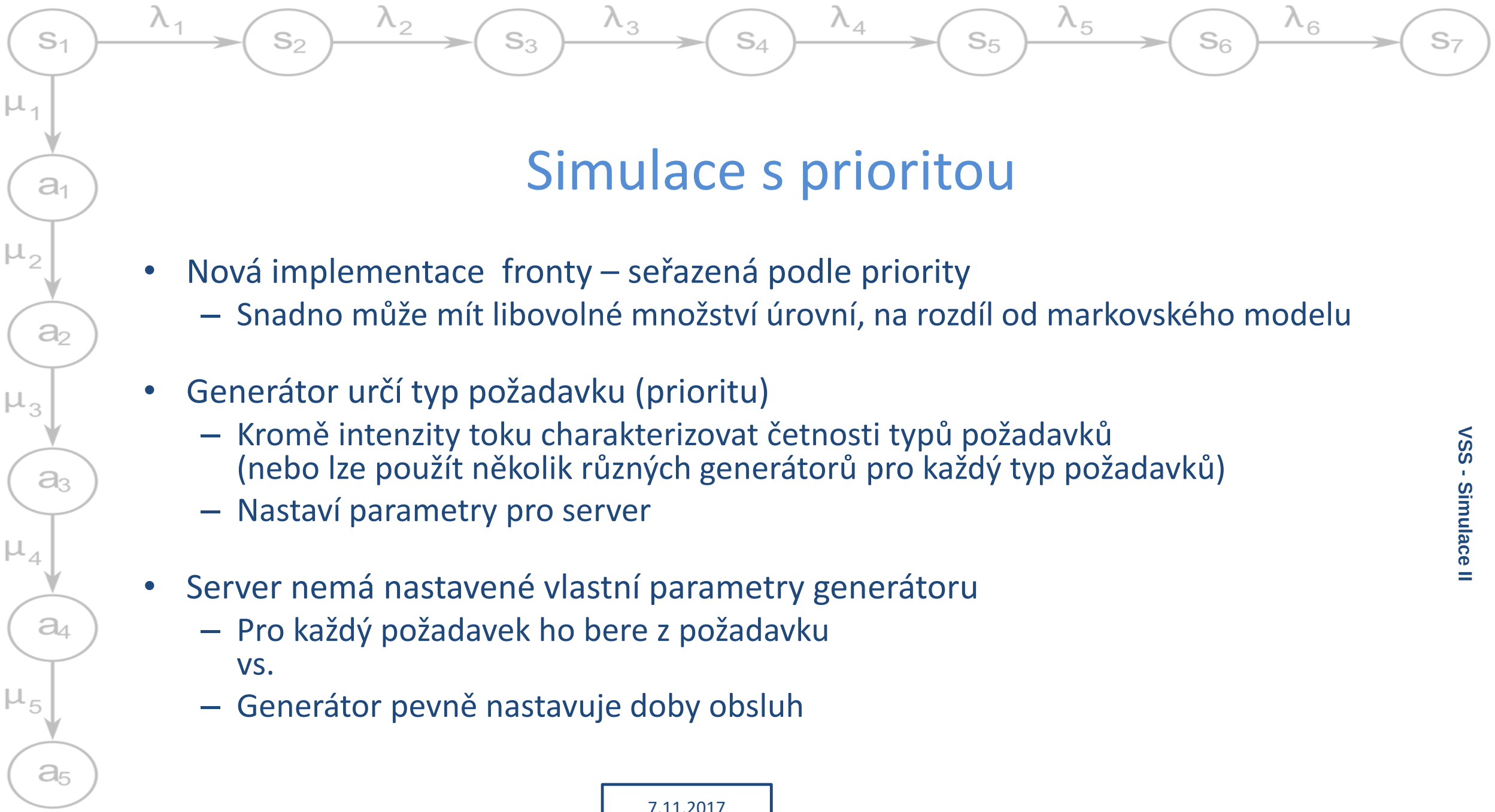
```
Link[] buffer = …
life() {
 while(true) {
  if(buffer.isEmpty()) {
   this.passivate();
  }
  buffer.remove();
  if(buffer.isEmpty() &&
    producer.isPassive()){
    producer.
     activate(rnd.timeλ());
  }
  hold(rnd.timeμ());
 }}
```

Složitější model – prioritní fronta

- 2 úrovně priority (rychlé a pomalé požadavky, pravděpodobnosti p_1, p_2), stejná priorita ve FIFO pořadí, 1 kanál obsluhy

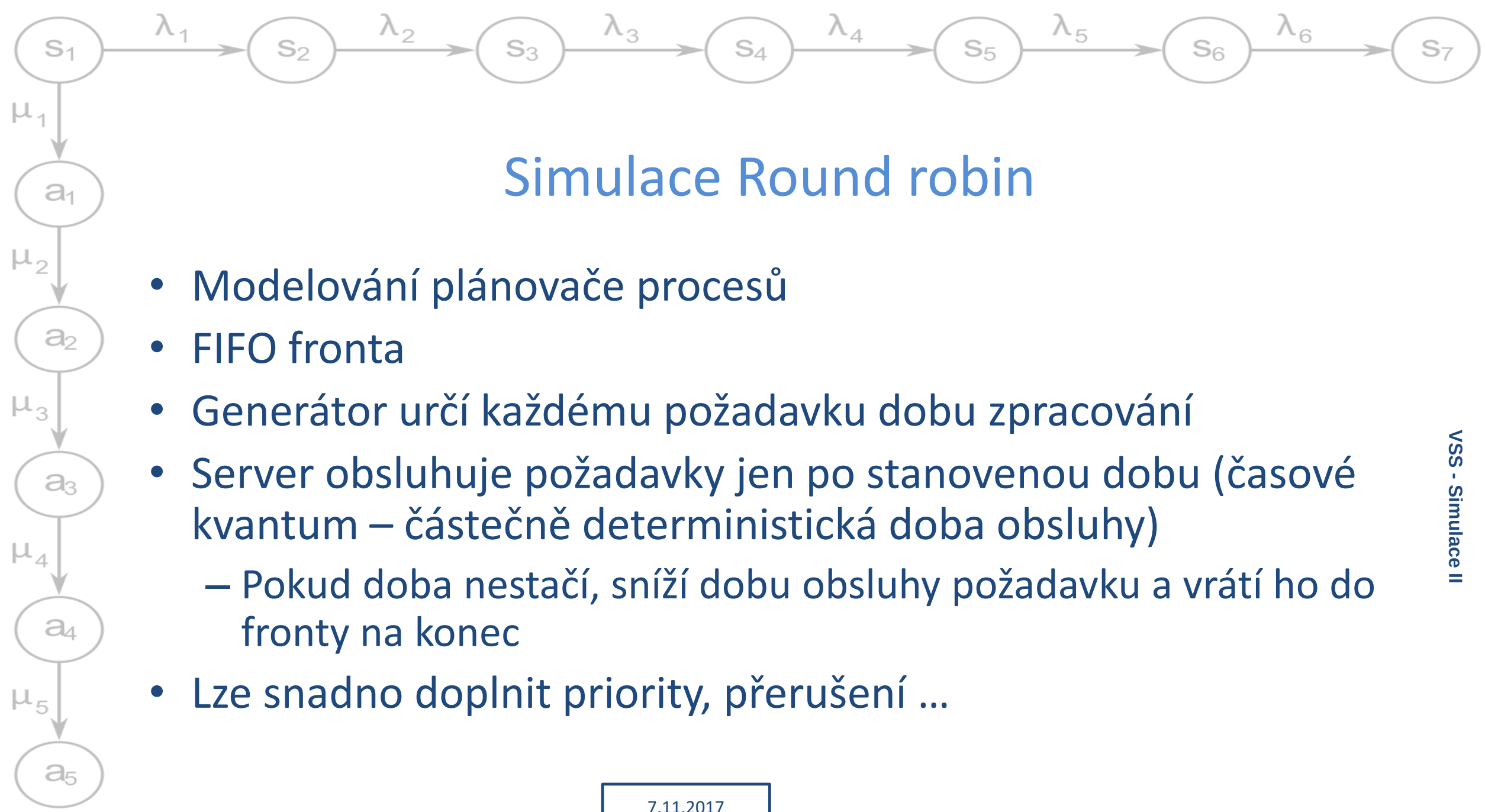
  → 2 rychlosti obsluhy (μ_1, μ_2) podle typu požadavku





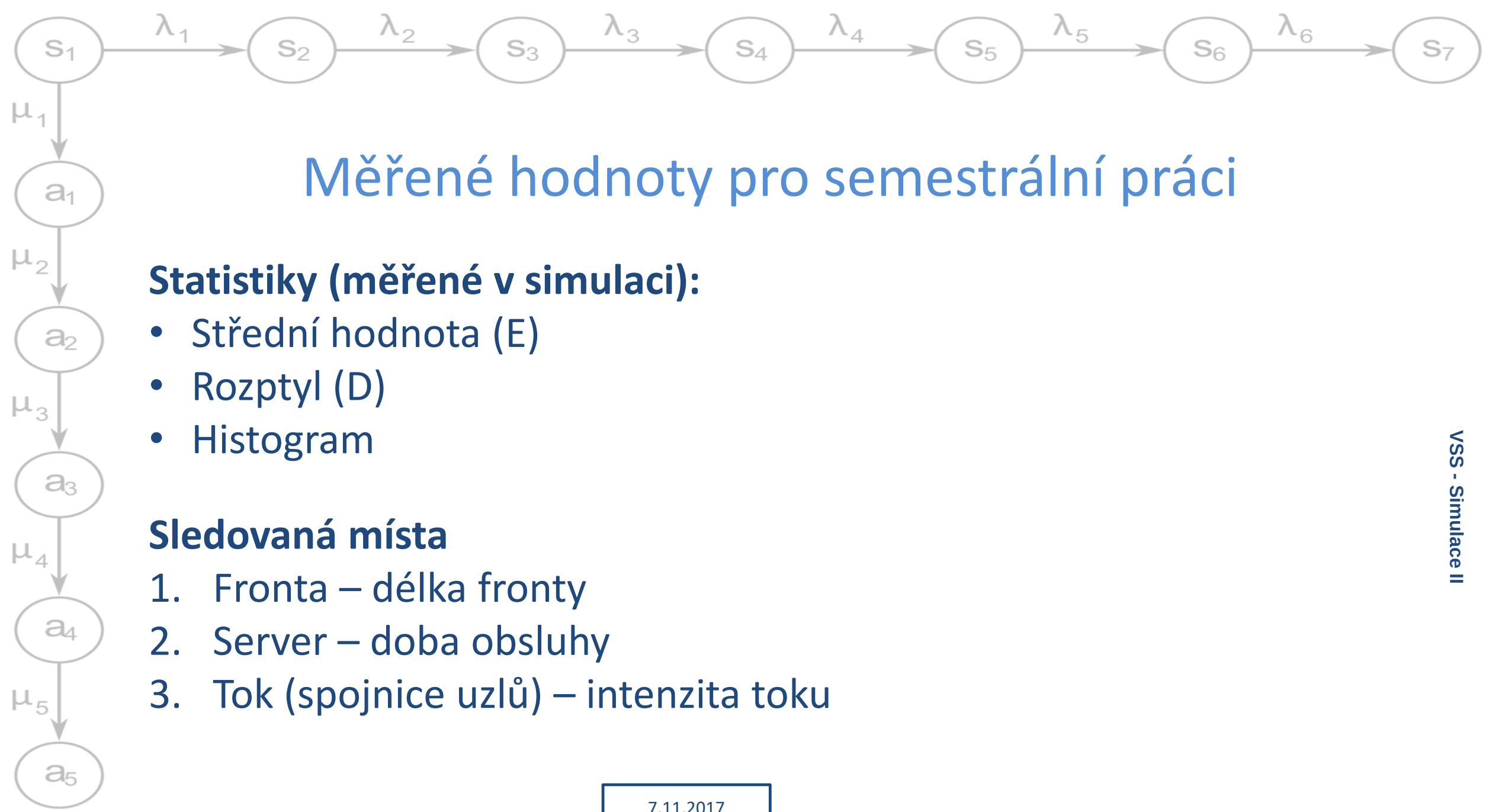
Simulace s prioritou

- Nová implementace fronty – seřazená podle priority
  - Snadno může mít libovolné množství úrovní, na rozdíl od markovského modelu
- Generátor určí typ požadavku (prioritu)
  - Kromě intenzity toku charakterizovat četnosti typů požadavků (nebo lze použít několik různých generátorů pro každý typ požadavků)
  - Nastaví parametry pro server
- Server nemá nastavené vlastní parametry generátoru
  - Pro každý požadavek ho bere z požadavku
    vs.
  - Generátor pevně nastavuje doby obsluh



Simulace Round robin

- Modelování plánovače procesů
- FIFO fronta
- Generátor určí každému požadavku dobu zpracování
- Server obsluhuje požadavky jen po stanovenou dobu (časové kvantum – částečně deterministická doba obsluhy)
  - Pokud doba nestačí, sníží dobu obsluhy požadavku a vrátí ho do fronty na konec
- Lze snadno doplnit priority, přerušení …



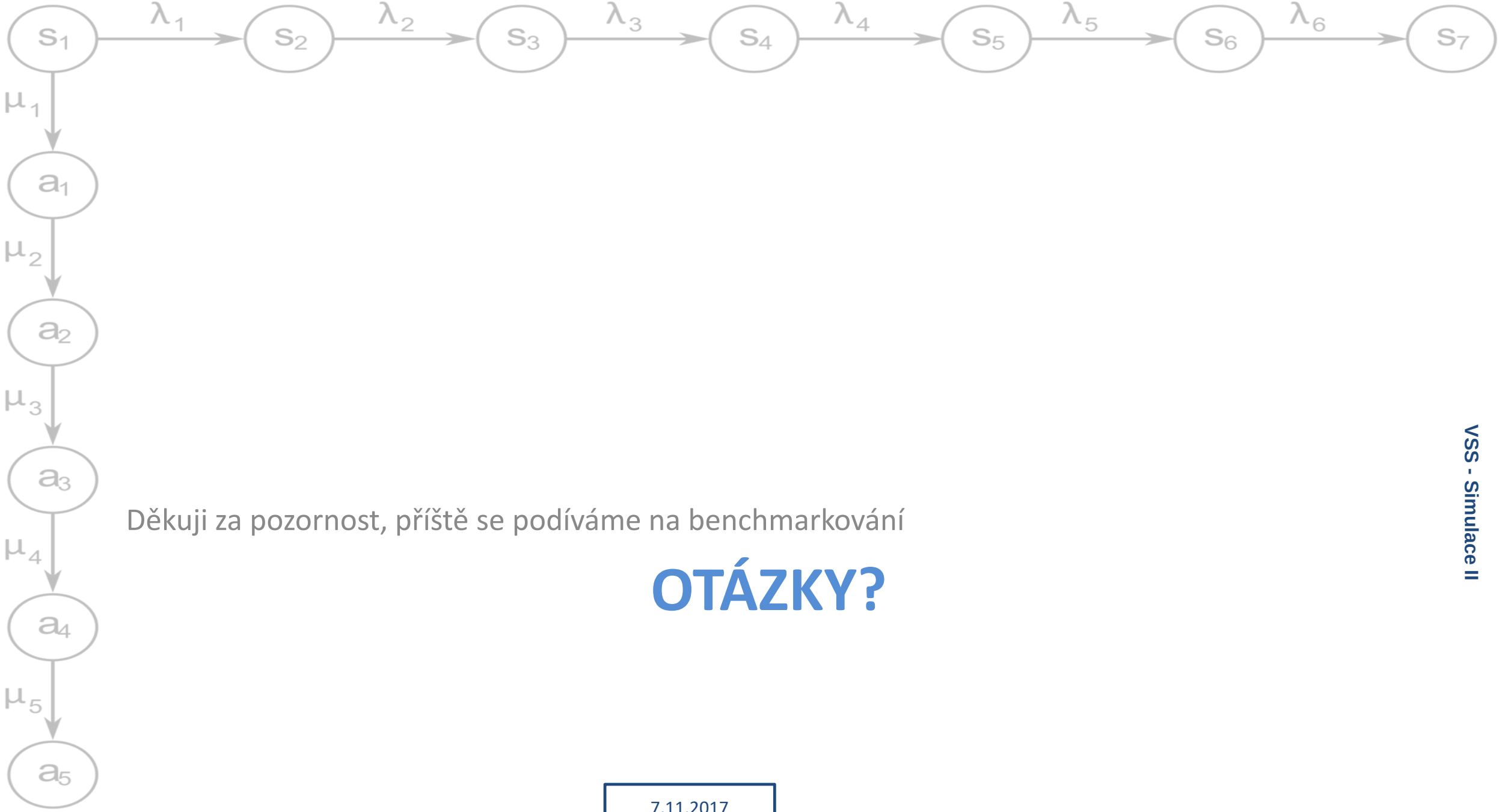
Měřené hodnoty pro semestrální práci

Statistiky (měřené v simulaci):

- Střední hodnota (E)
- Rozptyl (D)
- Histogram

Sledovaná místa

1. Fronta – délka fronty
2. Server – doba obsluhy
3. Tok (spojnice uzlů) – intenzita toku



Děkuji za pozornost, příště se podíváme na benchmarkování

OTÁZKY?