Симуляция, управляемая событиями

Курс «Программное моделирование вычислительных систем»

Григорий Речистов
grigory.rechistov@phystech.edu

27 марта 2015 г.

1 Таймер

2 Отложенный ответ

3 Теория

4 Практический пример

5 Косимуляция

6 Практический пример

7 Заключение

На прошлой лекции

- Модели ЦПУ — это интересно
- Но что делать, когда требуется модель более полной системы?

Вопросы

- В каких случаях можно использовать прямое исполнение для симуляции?

Вопросы

- В каких случаях можно использовать прямое исполнение для симуляции?
- Почему наивная схема не будет работать?

Вопросы

- В каких случаях можно использовать прямое исполнение для симуляции?
- Почему наивная схема не будет работать?
- Для чего ещё можно использовать механизм установки заглушек в целевом коде?

Пример №1: таймер

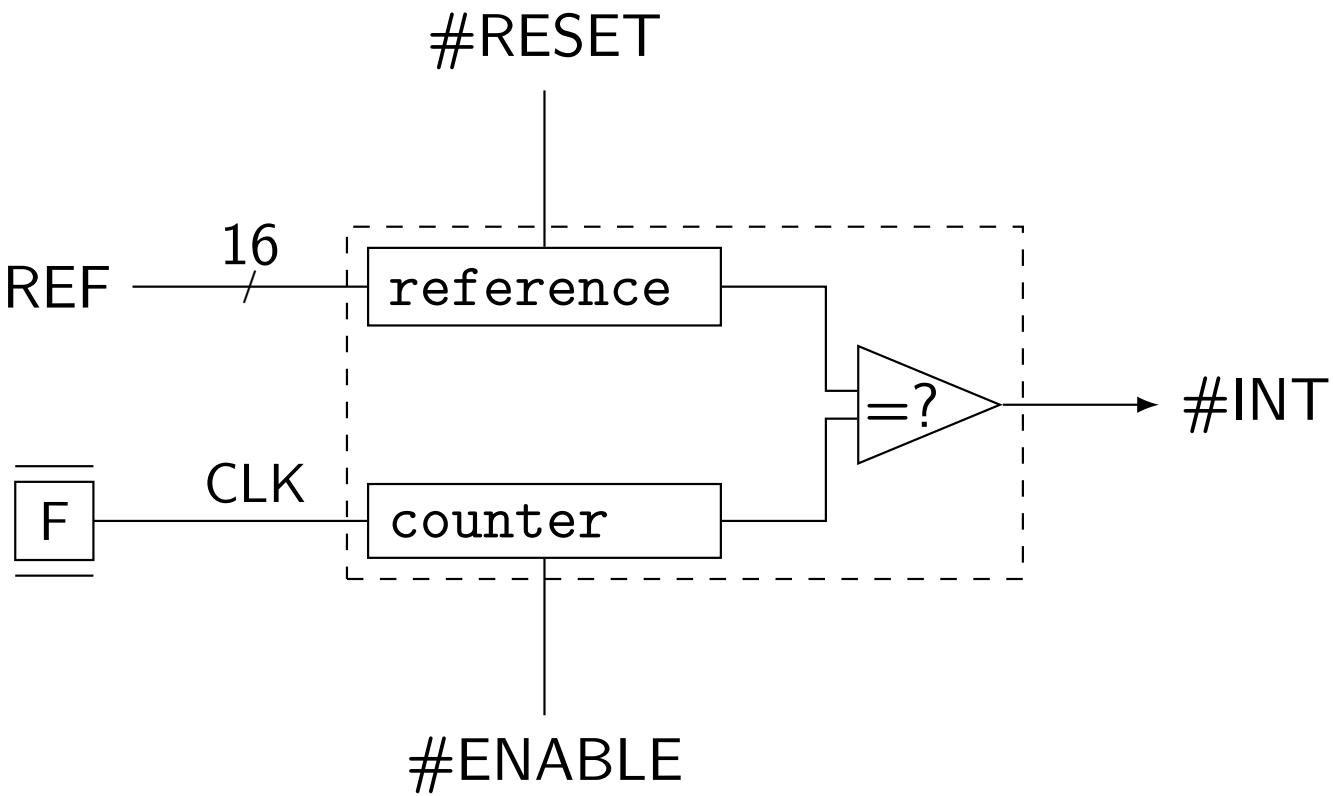
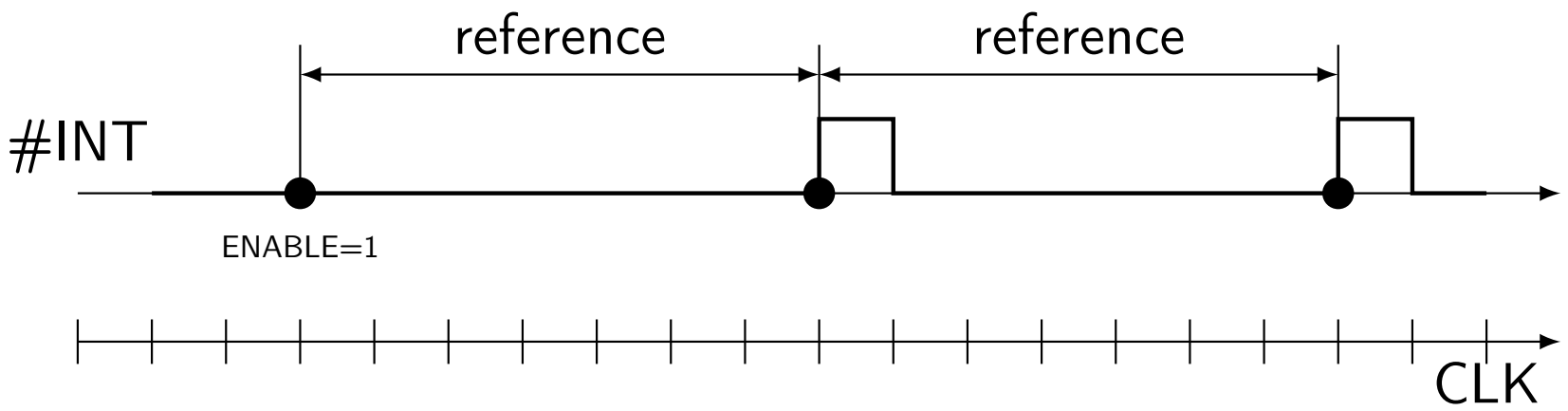


Диаграмма работы



Моделирование с фиксированным шагом

```
on_clk() {
  if (enable) counter +=1;
  if (counter == reference) {
      raise_int();
      counter = 0;
  } else {
      lower_int();
  }
}

on_reset() {
    reference = 0;
    counter = 0;
}
```

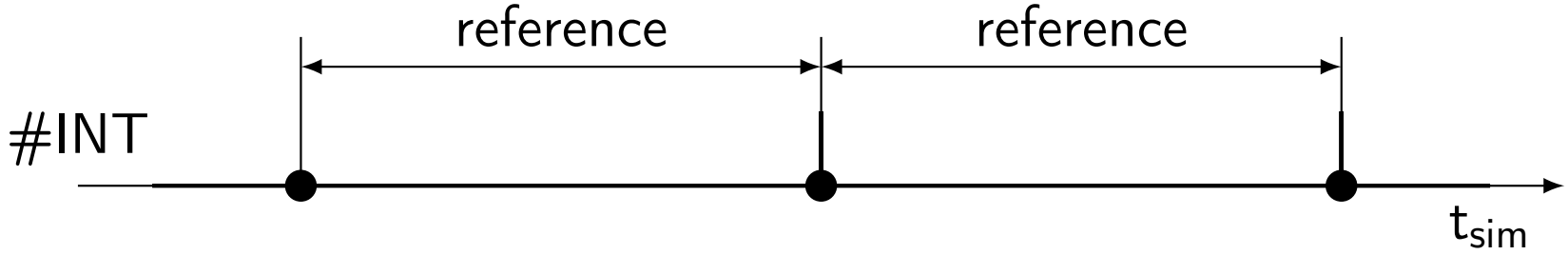
Типичные значения параметров таймера

- F ≈ 10 МГц
- reference $> 10^3$
- #RESET — не чаще одного раза в $\approx$ 100 секунд

$\Rightarrow$ внешне видимый эффект (#INT) происходит примерно один раз в 10^3 тактов.

Оптимизация

Не моделировать внешне ненаблюдаемые действия.

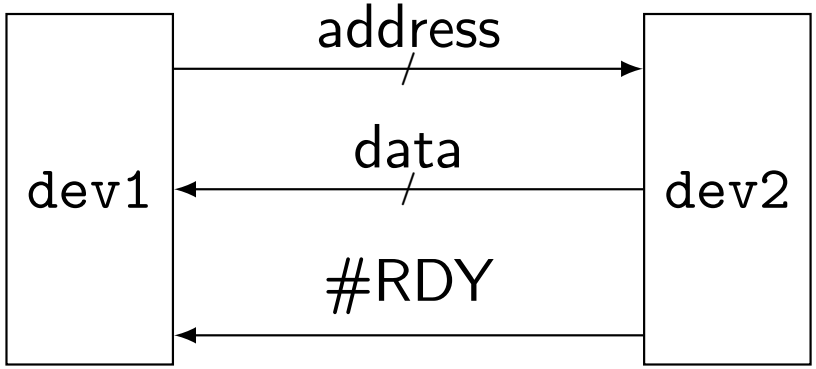


Моделирование событий

```
struct event_t {
    time_t delta;
    dev_t *device;
    (*function)();
}
event_t event_queue[100];
time = 0;
foreach e in event_queue {
    e.function(e.device);
    time +=e.delta;
}
```

Схема будет работать правильно для нескольких устройств сразу!

Пример №2: ожидание ответа



1. Запрос от `dev1`: address.
2. `dev2` вычисляет data.
3. `dev2` оповещает `dev1` о готовности данных *через некоторое время* ΔT с помощью #RDY.
4. `dev1` после отправки address и до получения #RDY работает независимо.

Реализация

`dev1`:

1. dev2.read(address)

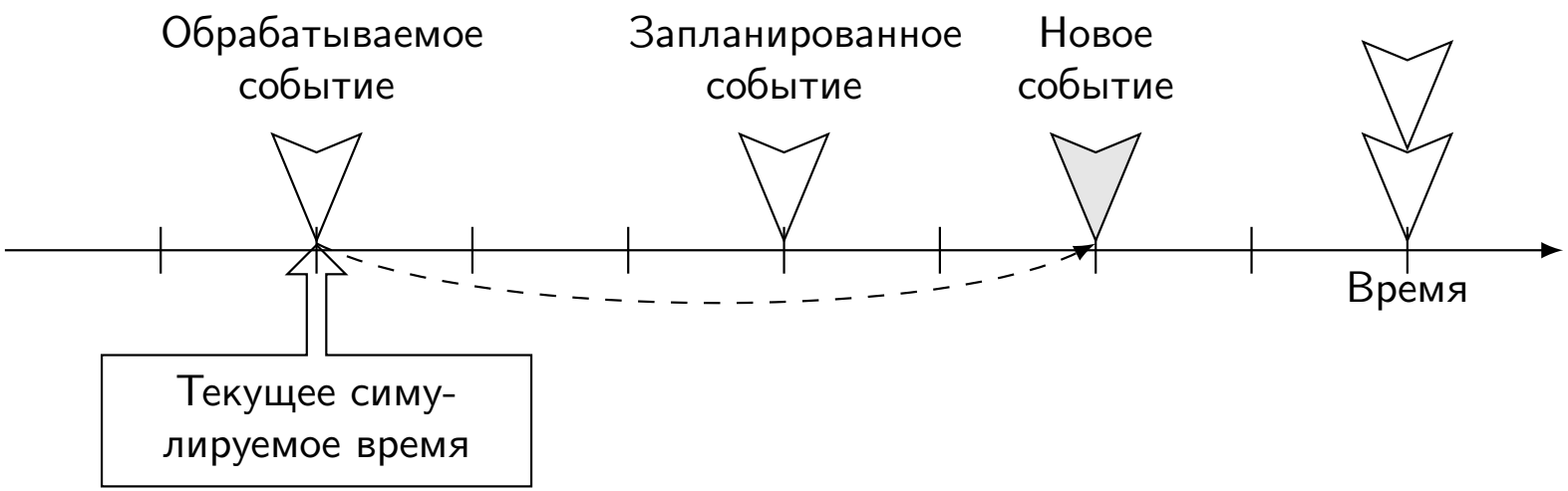
`dev2`:

1. data = get_data(address)
2. event_queue.post(ΔT, dev1, rdy())

`dev1`:

1. rdy(): чтение data

Очередь событий



Содержимое и результаты

Что содержится в одном событии:

- функция, которая должна быть вызвана,
- объект, состояние которого изменяется.

Результаты обработки события:

- изменение состояния системы,
- добавление/уничтожение событий.

Вопросы к модели

Что будет происходить с очередью событий при

1. выключении таймера (ENABLE $\leftarrow 0$)?

Вопросы к модели

Что будет происходить с очередью событий при

1. выключении таймера (ENABLE $\leftarrow$ 0)? Удаление неслучившихся событий из очереди
2. записи в `reference` или #RESET?

Вопросы к модели

Что будет происходить с очередью событий при

1. выключении таймера (ENABLE $\leftarrow$ 0)? Удаление неслучившихся событий из очереди
2. записи в `reference` или #RESET? Пересоздание событий в очереди
3. чтении регистра `counter`?

Алгоритм DES

```
struct event_t {
    time_t delta;
    dev_t *device;
    (*function)();
}
uint sim_time = 0;
while (! empty(queue)) {
    sim_time += get_delta(queue);
    evt_t evt = pop(queue);
    evt.fn(evt.device, queue);
}
```

Свойства событий в модели DES

- Порождаемые события не могут попасть в прошлое
- Обработка событий может не только порождать события в будущем, но и отменять некоторые из них (ещё не обработанные)
- Несколько событий могут иметь одинаковую метку времени

Пример на модели or1k

Демо

```
simics> log-level 1
New global log level: 1
simics> continue-cycles 199
[chip0] v:0x031c p:0x031c  nop
simics> peq
  Step  Object       Description

 Cycle  Object       Description
     1  tick0        reference_reached
499802  cosim_cell   sync_report
999801  sim          Time Quantum End
999801  cosim_cell   sync_block
```

Какие типы моделей нам известны

- Интерпретация

Какие типы моделей нам известны

- Интерпретация: процессоры

Какие типы моделей нам известны

- Интерпретация: процессоры
- Очередь событий

Какие типы моделей нам известны

- Интерпретация: процессоры
- Очередь событий: таймер

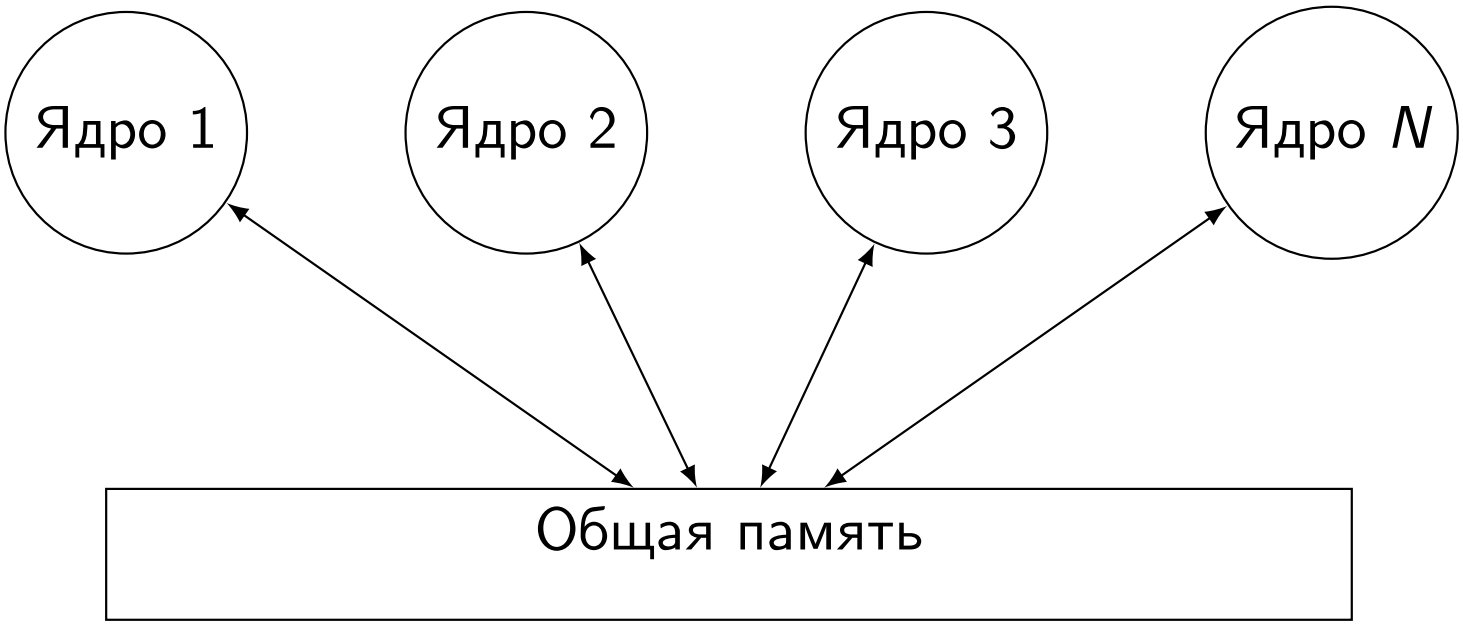
Какие типы моделей нам известны

- Интерпретация: процессоры
- Очередь событий: таймер
- «Мгновенная» модель «стимул — отклик»

Какие типы моделей нам известны

- Интерпретация: процессоры
- Очередь событий: таймер
- «Мгновенная» модель «стимул — отклик»: ОЗУ

Модель многопроцессорной системы



Step-by-step

- Как выдержать одновременность симуляции инструкций на всех гостевых процессорах? Исполнять не более одной инструкции за раз

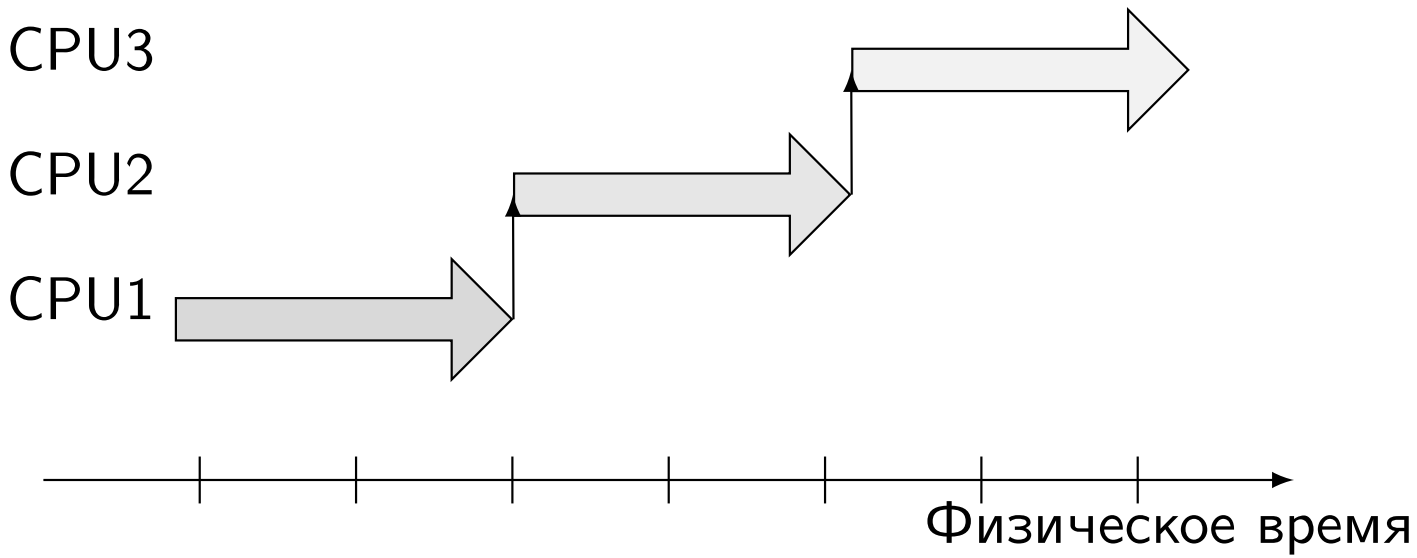
Step-by-step

- Как выдержать одновременность симуляции инструкций на всех гостевых процессорах? Исполнять не более одной инструкции за раз
- Но это будет очень медленно! Может быть, возможно исполнять несколько гостевых инструкций без переключения?

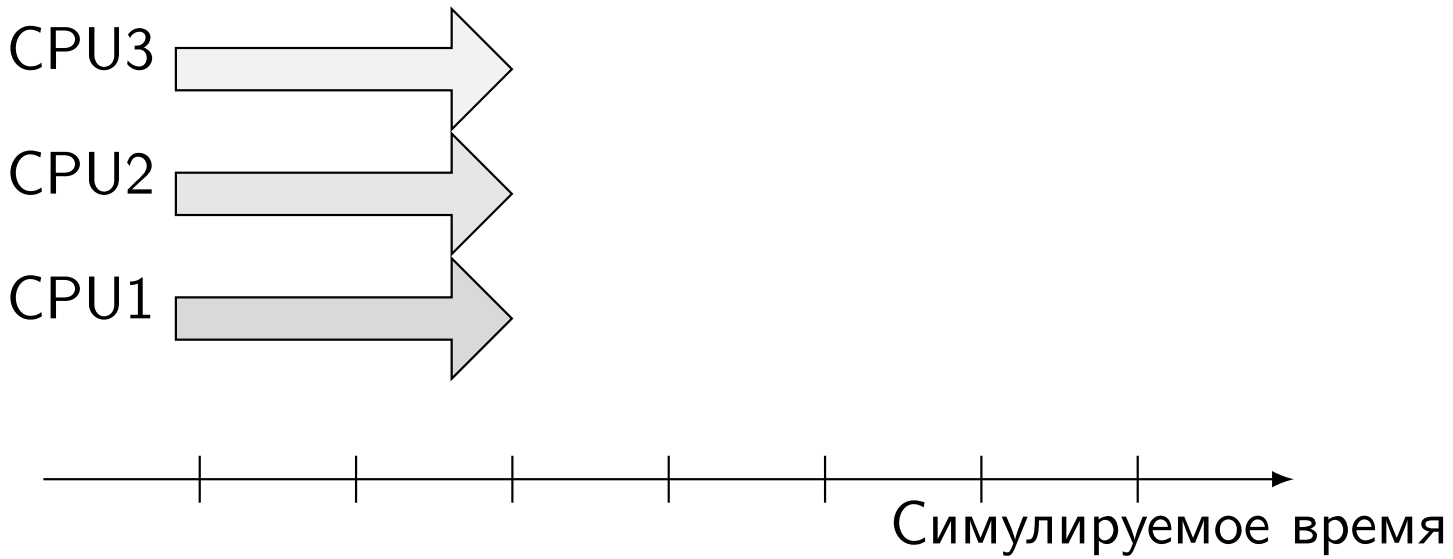
Step-by-step

- Как выдержать одновременность симуляции инструкций на всех гостевых процессорах? Исполнять не более одной инструкции за раз
- Но это будет очень медленно! Может быть, возможно исполнять несколько гостевых инструкций без переключения?
- А каково приблизительное время доставки сигналов от одного процессора до другого в реальности?

Квотированная симуляция



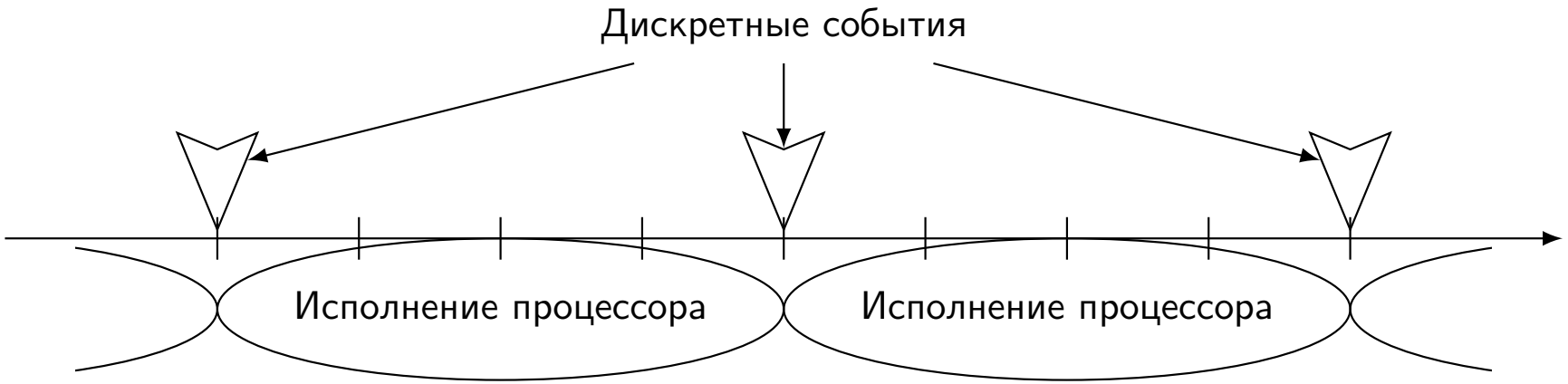
Квотированная симуляция



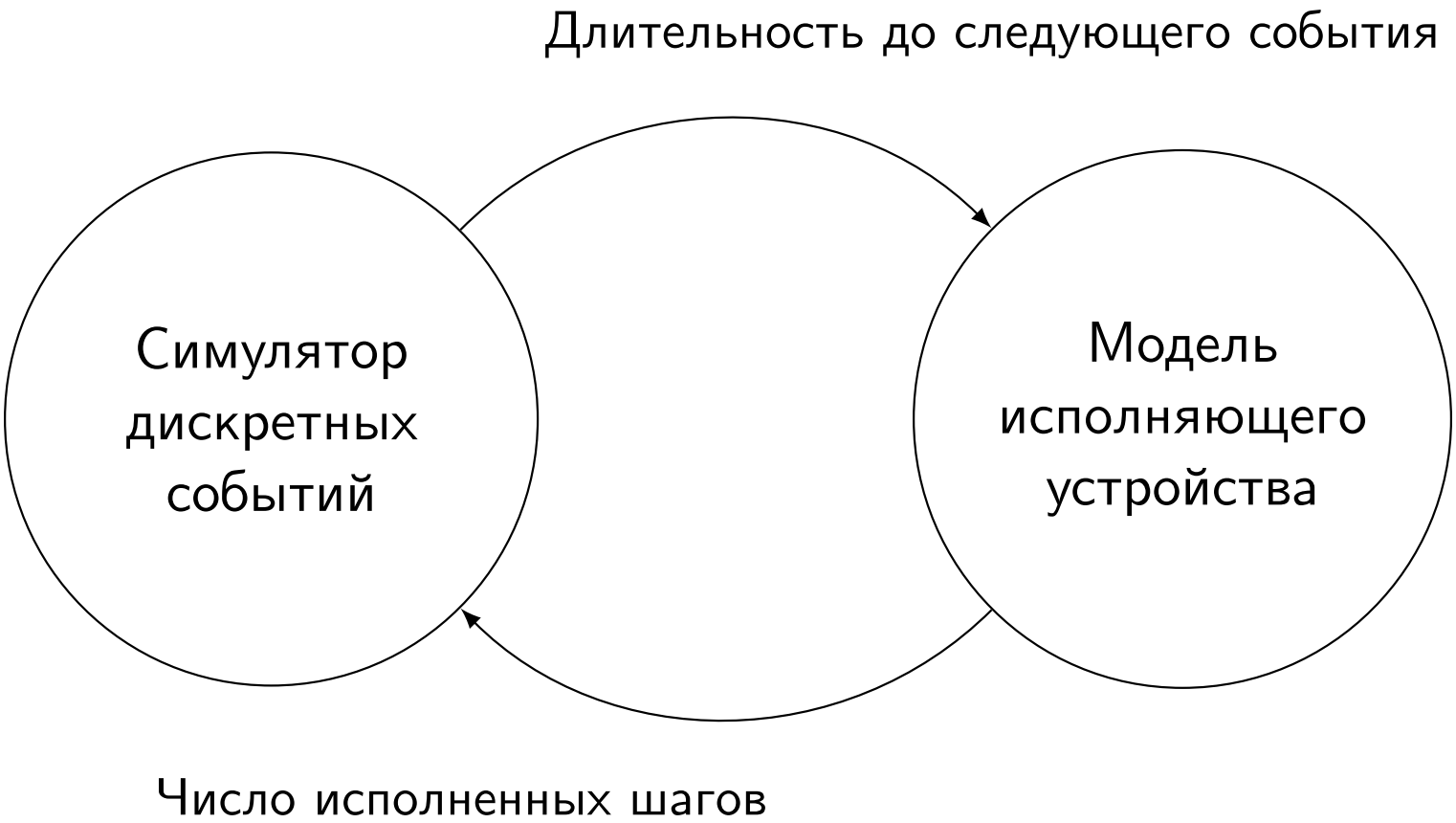
Размер квоты

- Процессор может исполнить меньше инструкций, чем содержится в выданной ему квоте
- Не следует увлекаться излишне большими квотами, пытаясь ускорить исполнение
- В DES-модели могут быть реализованы псевдо-события, обработка которых вызывает переключение текущего процессора

Совместная симуляция DES и исполняющей модели



Косимуляция



Пример на модели viper-busybox.simics

Квота на восьмиядерной конфигурации:

```
simics> cpu-switch-time
Current time quantum:      0.0001 s
  200000.0  viper.mb.cpu0.core[0][0]
  200000.0  viper.mb.cpu0.core[0][1]
  200000.0  viper.mb.cpu0.core[1][0]
  200000.0  viper.mb.cpu0.core[1][1]
  200000.0  viper.mb.cpu0.core[2][0]
  200000.0  viper.mb.cpu0.core[2][1]
  200000.0  viper.mb.cpu0.core[3][0]
  200000.0  viper.mb.cpu0.core[3][1]
Default time quantum not set yet
```

Пример на модели viper-busybox.simics

Симулируемое время на восьмиядерной конфигурации:

```
simics> ptime -all
processor                        steps       cycles  time [s]
viper.mb.cpu0.core[0][0]  4602030000  5134030000    2.567
viper.mb.cpu0.core[0][1]  4925600000  5134000000    2.567
viper.mb.cpu0.core[1][0]  4925600000  5134000000    2.567
viper.mb.cpu0.core[1][1]  4925600000  5134000000    2.567
viper.mb.cpu0.core[2][0]  4925600000  5134000000    2.567
viper.mb.cpu0.core[2][1]  4925600000  5134000000    2.567
viper.mb.cpu0.core[3][0]  4925600000  5134000000    2.567
viper.mb.cpu0.core[3][1]  4925600000  5134000000    2.567
```

Итоги

- Моделирование с фиксированным шагом
- Моделирование, управляемое событиями
- Создание, обработка и удаление событий
- Квотированная симуляция исполняющих моделей
- Взаимодействие исполняющих и неисполняющих моделей

Литература I


Handbook of Simulation. Principles, Methodology, Advances, Applications, and Practice / ed. by J. Banks. — John Wiley & Sons, Inc., 1998. — ISBN 0-471-13403-1. — http://books.google.com/books?id=dMZ1Zj3TBgAC

Спасибо за внимание!

Слайды и материалы курса доступны по адресу
http://is.gd/ivuboc

Замечание: все торговые марки и логотипы, использованные в данном материале, являются собственностью их владельцев. Представленная здесь точка зрения отражает личное мнение автора, не выступающего от лица какой-либо организации.