Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Операционные системы: *аспекты параллелизма*

Взаимоблокировка

Линёв А.В.

2007

Тема обсуждения

- Операционная система управляет многими видами ресурсов
- Если процессам предоставляются исключительные права доступа к ресурсам, то процесс, получивший в свое распоряжение один ресурс и затребовавший другой, может ожидать предоставления второго ресурса в течение неопределенного времени

Взаимоблокировка (Тупик)

- Существуют неразделяемые ресурсы
  - одиночные или счетные (то есть имеющиеся более чем в одном экземпляре)
  - аппаратные или программные: принтеры, ленточные накопители, центральный процессор, файлы и записи в них, семафоры и т.д.
- Работа процесса с ресурсами включает три стадии
  - Получить/захватить ресурс
    - если ресурс свободен, он предоставляется процессу
    - если ресурс занят, процесс блокируется
  - Использовать ресурс
  - Освободить ресурс
- Возможна следующая нежелательная ситуация
  - Процесс А захватил ресурс 1 и ожидает предоставления ему ресурса 2
  - Процесс В захватил ресурс 2 и ожидает предоставления ему ресурса 1
- Такая ситуация называется ***взаимоблокировкой*** или ***тупиком*** (deadlock)

Взаимоблокировка (Тупик). Пример

```
Semaphore  Sem1 = 1; /* управляет доступом к ресурсу 1 */
Semaphore  Sem2 = 1; /* управляет доступом к ресурсу 2 */
```

```
Функция процесса A:
{
   /* инициализация */

1  P(Sem1);
4  P(Sem2);

   /* использование
      обоих ресурсов */

   V(Sem2);
   V(Sem1);
}
```

```
Функция процесса B:
{
   /* инициализация */

2  P(Sem2);
3  P(Sem1);

   /* использование
      обоих ресурсов */

   V(Sem2);
   V(Sem1);
}
```

После шага 4 мы получили взаимоблокировку

Взаимоблокировка (Тупик). Определение

- Множество процессов находится в тупиковой ситуации, если
  - каждый процесс ожидает некоторого события
  - это событие может быть вызвано только действиями другого процесса из данного множества

- В большинстве случаев ожидаемое событие – освобождение некоторого ресурса. В дальнейшем рассмотрении мы будем предполагать именно такую ситуацию

Взаимоблокировка (Тупик). Необходимые условия

- Для возникновения ситуации взаимоблокировки должны выполняться следующие условия:
  - **Mutual Exclusion** (**Взаимное исключение**) – по крайней мере один из запрашиваемых ресурсов является неделимым (то есть должен захватываться в эксклюзивное использование)
  - **Hold and wait** (**Удержание ресурсов при ожидании**) – существует процесс, владеющий некоторым ресурсом и ожидающий освобождения другого ресурса
  - **No preemption** (**Неперераспределяемость ресурсов**) – ресурсы не могут быть отобраны у процесса без его желания
  - **Circular wait** (**Циклическое ожидание**) – существует такое множество процессов $\{P_1, P_2,\ldots, P_N\}$, в котором P_1 ждет P_2, P_2 ждет $P_3,\ldots, P_N$ ждет P_1

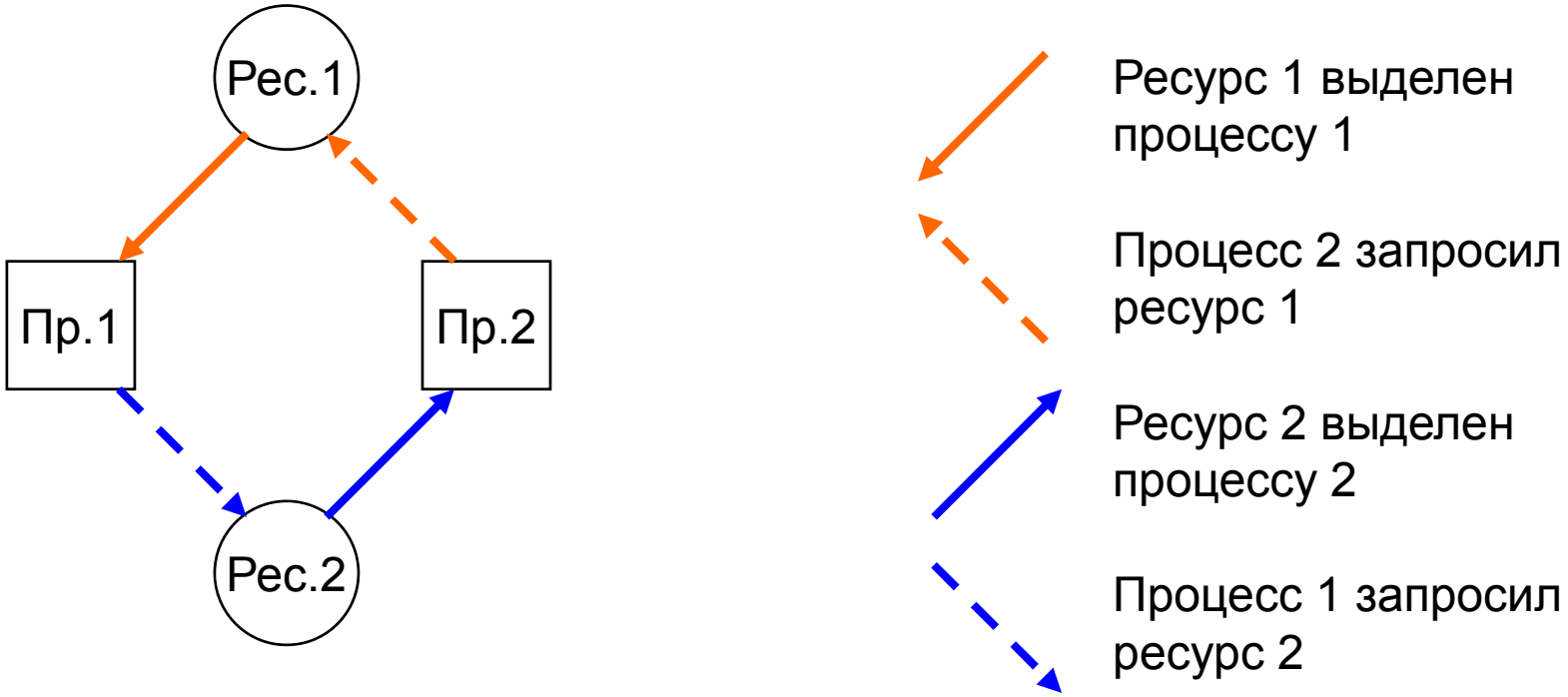
Коффман, Элфик, Шошани
1971 г.

Граф "Процесс-Ресурс"

Граф "Процесс-Ресурс". Определение

- Направленный граф "процесс-ресурс" включает:
  - Множество вершин $V = P \cup R$, где $P=\{P_1,P_2,\ldots,P_N\}$ – множество процессов, $R=\{R_1,R_2,\ldots,R_M\}$ – множество ресурсов
  - Дуги запросов – направлены от процессов к ресурсам дуга $P_i \rightarrow R_j$ означает, что процесс P_i запросил ресурс R_j
  - Дуги распределения – направлены от ресурсов к процессам дуга $R_j \rightarrow P_i$ означает, что ресурс R_j выделен процессу P_i
- Если граф "процесс-ресурс" не имеет циклов, тупиков нет
- Если граф "процесс-ресурс" имеет цикл, возможно, тупик существует

Граф "Процесс-Ресурс". Пример



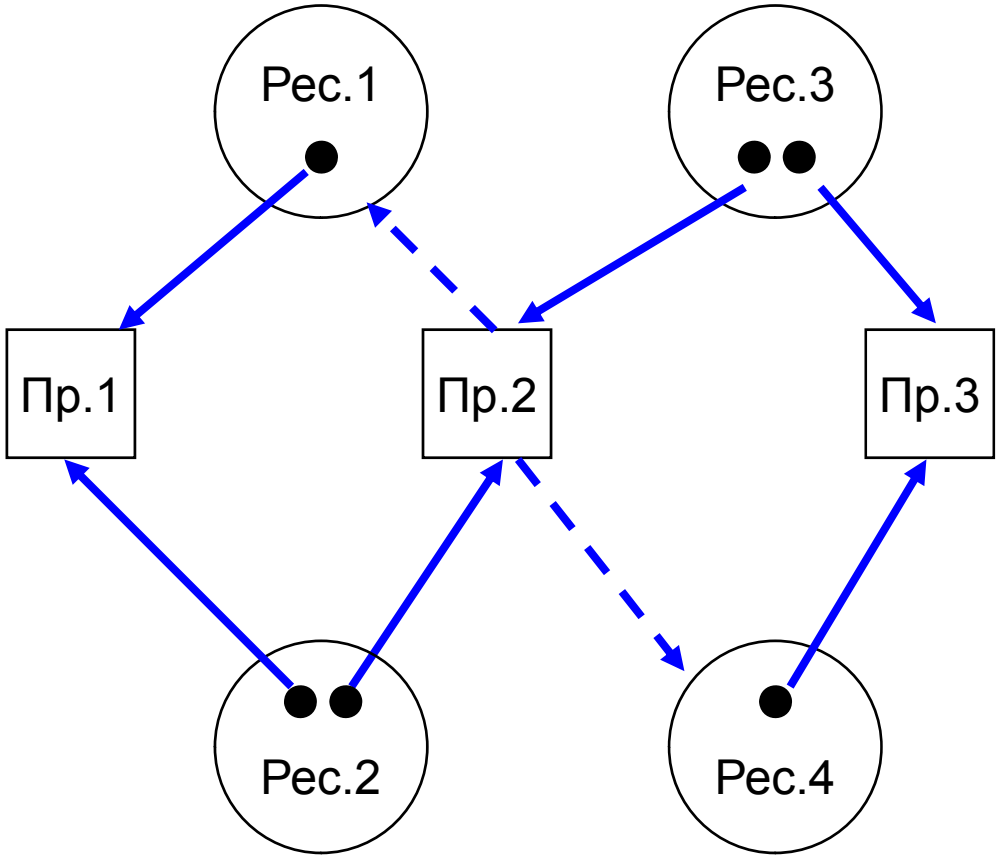
Граф "Процесс-Ресурс". Редукция графа

- Граф "процесс-ресурс" может быть сокращен за счет процесса, все запросы которого могут быть удовлетворены
- При сокращении все ресурсы выбранного процесса освобождается – все дуги графа, обозначающие выделения ресурсов этому процессу, удаляются

Доказан ряд утверждений, связанных с процессом редукции

- Если граф полностью редуцируем, взаимоблокировка отсутствует
- Порядок выполнения редукции не имеет значения

Граф "Процесс-Ресурс". Пример

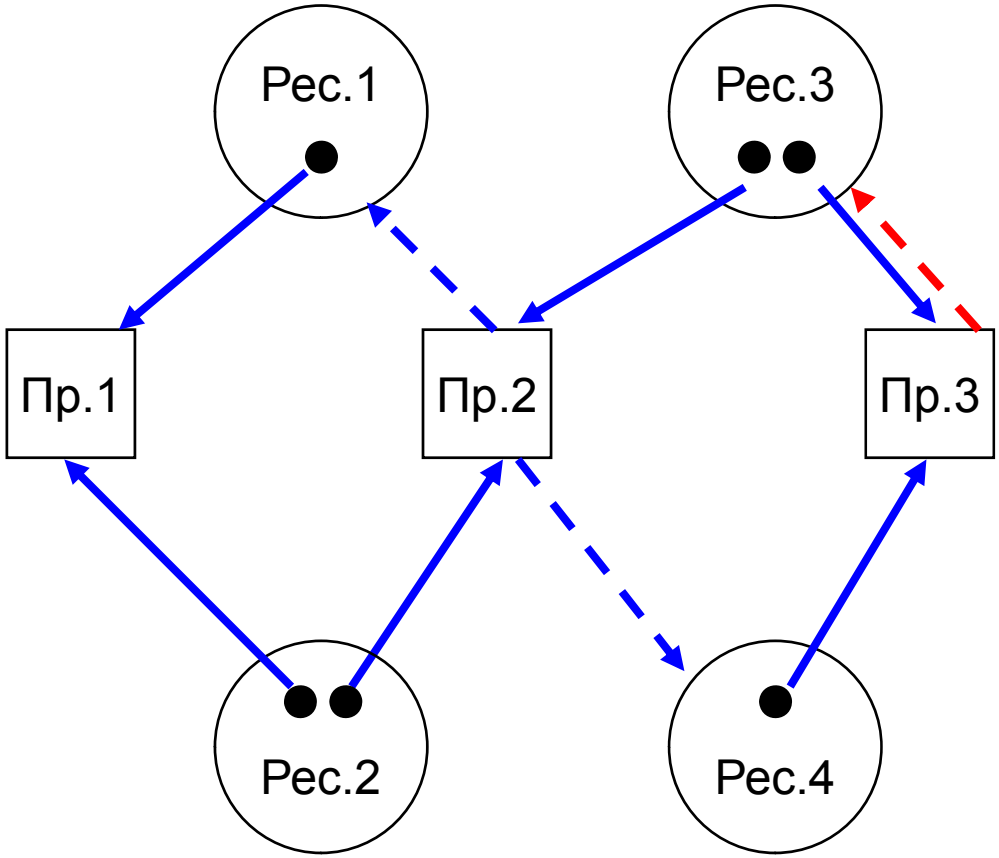


Ресурс выделен процессу

Процесс запросил ресурс

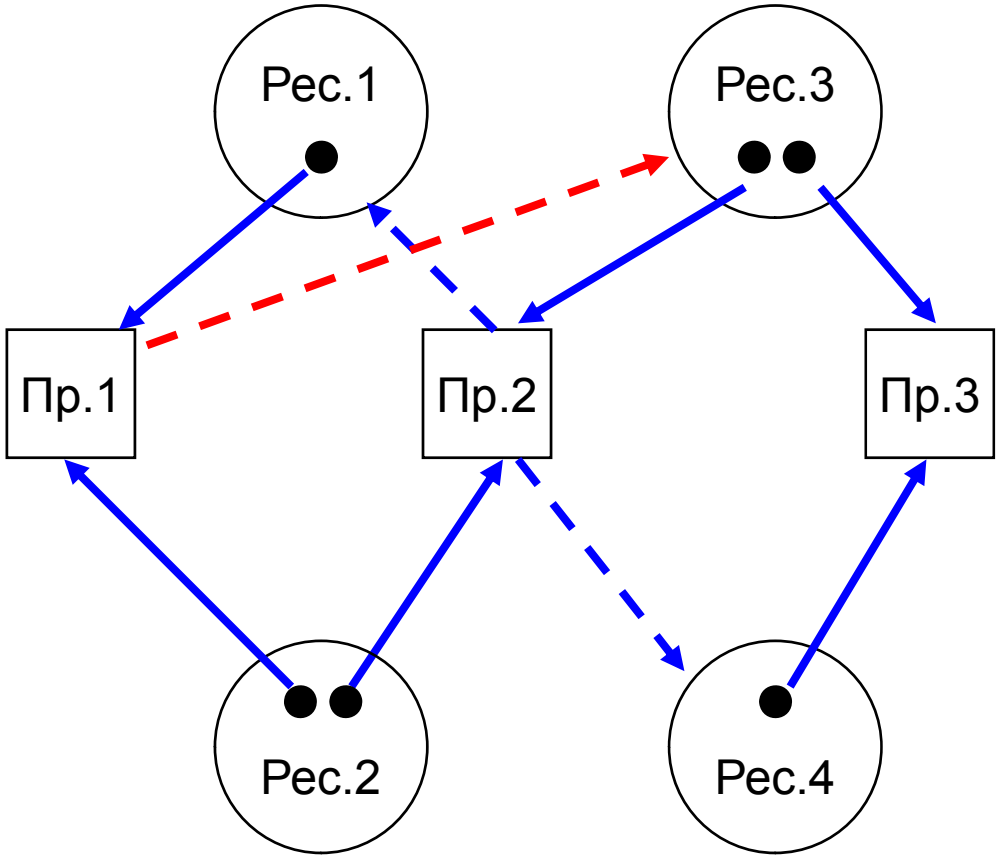
Какие запросы могут привести к возникновению взаимоблокировки?

Граф "Процесс-Ресурс" Пример



Есть взаимоблокировка

Граф "Процесс-Ресурс" Пример



Нет взаимоблокировки

Взаимоблокировка (тупик). Подходы к решению проблемы

- Игнорирование проблемы ("Алгоритм страуса") – считаем, что тупики никогда не возникают, либо возникают достаточно редко и не приносят значительного ущерба
  - "За производительность и удобство работы пользователей стоит заплатить такую цену, как нечастые сбои в работе"
- Предупреждение тупиков
  - Предотвращение тупиков (prevention) – достигается устранением одного из 4 условий существования тупика (не допускает возникновения тупиков)
  - Избегание тупиков (avoidance) – аккуратное распределение ресурсов на основании имеющейся информации об их планируемом использовании позволяет избежать возникновения тупиков
- Устранение тупиков – позволяем тупику возникнуть, затем обнаруживаем и устраняем его. Включает две стадии: обнаружение тупика и восстановление после возникновения тупика (detection & recovery)

Предотвращение тупиков

Предотвращение тупиков. Устранение взаимоисключения

- Устранения условия "Mutual exclusion" (взаимное исключение) можно достигнуть, построив над ресурсом абстракцию, позволяющую использовать ресурс нескольким процессам одновременно
  - Пример: абстракция "очередь печати", построенная над ресурсом "принтер"
- К сожалению, это далеко не всегда возможно и требует дополнительных ресурсов

Предотвращение тупиков. Устранение неперераспределяемости

- Для устранения условия "No preemption" (Неперераспределяемость ресурсов) требуется обеспечить выполнение следующих операций
  - запоминание контекста работы процесса с ресурсом
  - передача ресурса другому процессу
  - возврат ресурса процессу и восстановление контекста работы процесса с ресурсом
- Для каких-то ресурсов это возможно (например, для ЦП), для других – нет (например, для принтера)
- Момент передачи ресурсов (2 подхода)
  - Если процесс затребовал ресурсы, часть из которых недоступна, перераспределяем принадлежащие ему ресурсы
  - Перераспределяем ресурсы процессов в состоянии ожидания для удовлетворения поступившего запроса от процесса в состоянии выполнения

Предотвращение тупиков. Устранение условия Hold&Wait

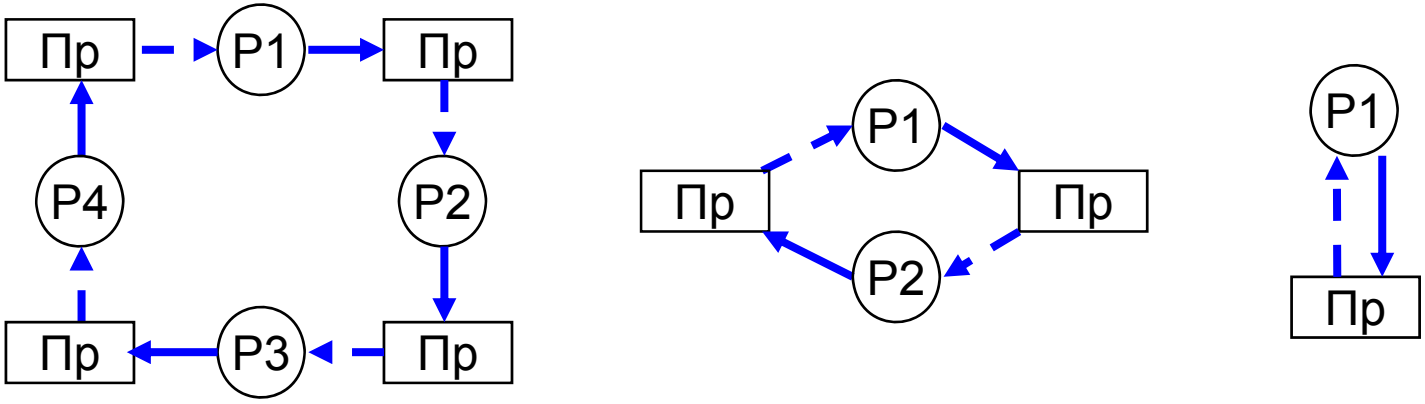
- Исключение данного условия – задача прикладных программистов. Существует несколько подходов:
  - Можно запрашивать все необходимые ресурсы до начала выполнения
    - Но обычно процессы не знают, какие ресурсы им понадобятся
    - Возможно голодание при ожидании множества популярных ресурсов
    - Ресурсы используются неэффективно (возможно, часть ресурсов нужна на очень короткое время)
  - При возникновении потребности в дополнительных ресурсах – освобождаем все уже имеющиеся, затем запрашиваем те, что необходимы в настоящий момент
    - Также возможно голодание и имеет место неэффективное использование ресурсов
- Как правило, в ОС имеются вызовы, позволяющие прикладной программе реализовать алгоритм, не оставляющий за собой владение уже имеющимися ресурсами при запросе дополнительных

Предотвращение тупиков. Устранение условия Circular wait…

- Можно позволить процессам владеть только одним ресурсом в каждый момент времени
- Можно ввести для ресурсов нумерацию и обязать процессы запрашивать ресурсы строго в порядке возрастания их номеров

Предотвращение тупиков. Устранение условия Circular wait

- Ресурсам присваиваются номера (P1, P2,…)
- Позволяется запрашивать ресурсы строго в порядке возрастания (или убывания) их номеров
- Идея: в цикле всегда есть процесс, у которого номер имеющегося ресурса больше номера запрошенного (исключение составляет случай, когда процесс запрашивает еще одну единицу уже имеющегося у него ресурса)



- Недостатки подхода
  - Нумерация не всегда возможна
  - Неэффективное использование ресурсов

Избегание тупиков

Избегание тупиков. Алгоритм Банкира

- Если у нас имеется информация о будущем, можем ли мы гарантировать, что тупик не возникнет?
  - Считаем, что максимальные требования всех процессов к ресурсам известны до начала выполнения процессов
- Стратегия избегания тупиков, Алгоритм Банкира:
  - Перед выделением ресурса проверяем, является ли состояние, в которое мы перейдем после выделения, безопасным
  - Если новое состояние безопасно – выделяем ресурс
  - Если новое состояние небезопасно – ресурс не выделяем, блокируем процесс, выполнивший запрос

Избегание тупиков. Безопасное состояние

- Состояние называется **безопасным**, если для него имеется последовательность процессов $\{P_1, P_2, \ldots, P_n\}$ такая, что для каждого P_i ресурсы, которые затребовал P_i, могут быть предоставлены за счет имеющихся незанятых ресурсов и ресурсов, выделенных всем процессам P_j, где $j < i$
- Состояние безопасно, поскольку ОС может гарантированно избежать тупика посредством блокирования любых новых запросов, пока не выполнится безопасная последовательность
- Данная стратегия основана на идее избегания возникновения циклического ожидания

Избегание тупиков
Безопасное состояние

- Предположим, имеются 12 устройств хранения

Процесс	MAX потребность	Владеет	Может запросить
p0	10	5	5
p1	4	2	2
p2	9	2	7

3 устройства свободны

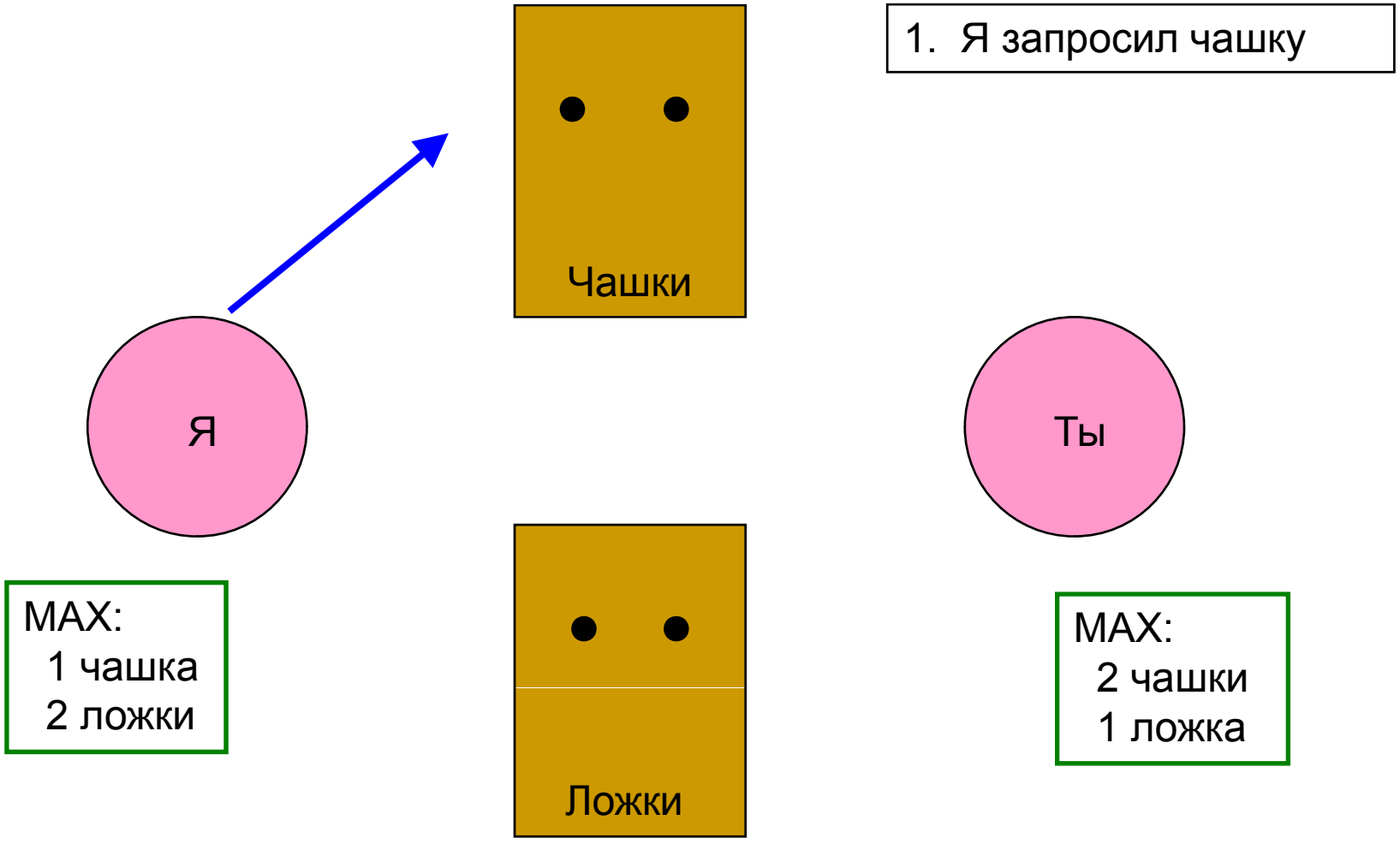
- Текущее состояние безопасно, поскольку существует безопасная последовательность: $<p_1,p_0,p_2>$
  - p_1 может завершиться, используя только свободные ресурсы
  - p_0 может завершиться, используя свободные ресурсы + ресурсы, которые сейчас выделены процессу p_1
  - p_2 может завершиться, используя свободные ресурсы + ресурсы, которые сейчас выделены процессам p_1 и p_0

- Если p_2 запросит еще 1 устройство, удовлетворение запроса будет отложено, поскольку оно переведет систему в небезопасное состояние

Избегание тупиков. Граф "Процесс-Ресурс"

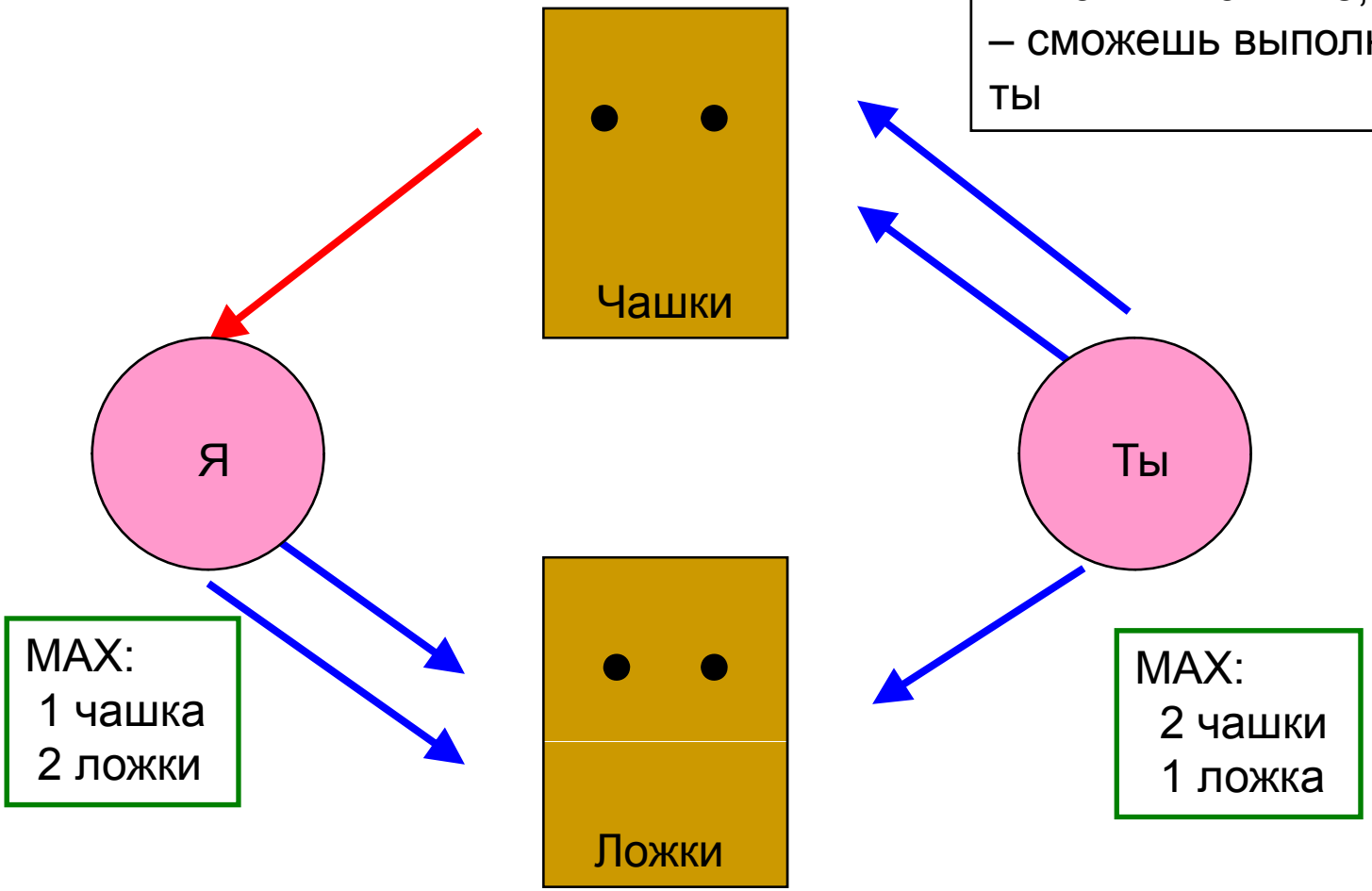
- Алгоритм банкира можно проиллюстрировать, используя граф "процесс-ресурс"
- При выполнении запроса:
  - представляем, что мы его удовлетворили
  - представляем, что все остальные возможные запросы удовлетворены
  - проверяем, может ли граф "процесс-ресурс" быть полностью редуцирован?
  - если да – выделяем запрошенный ресурс
  - если нет – блокируем процесс, выполнивший запрос

Избегание тупиков

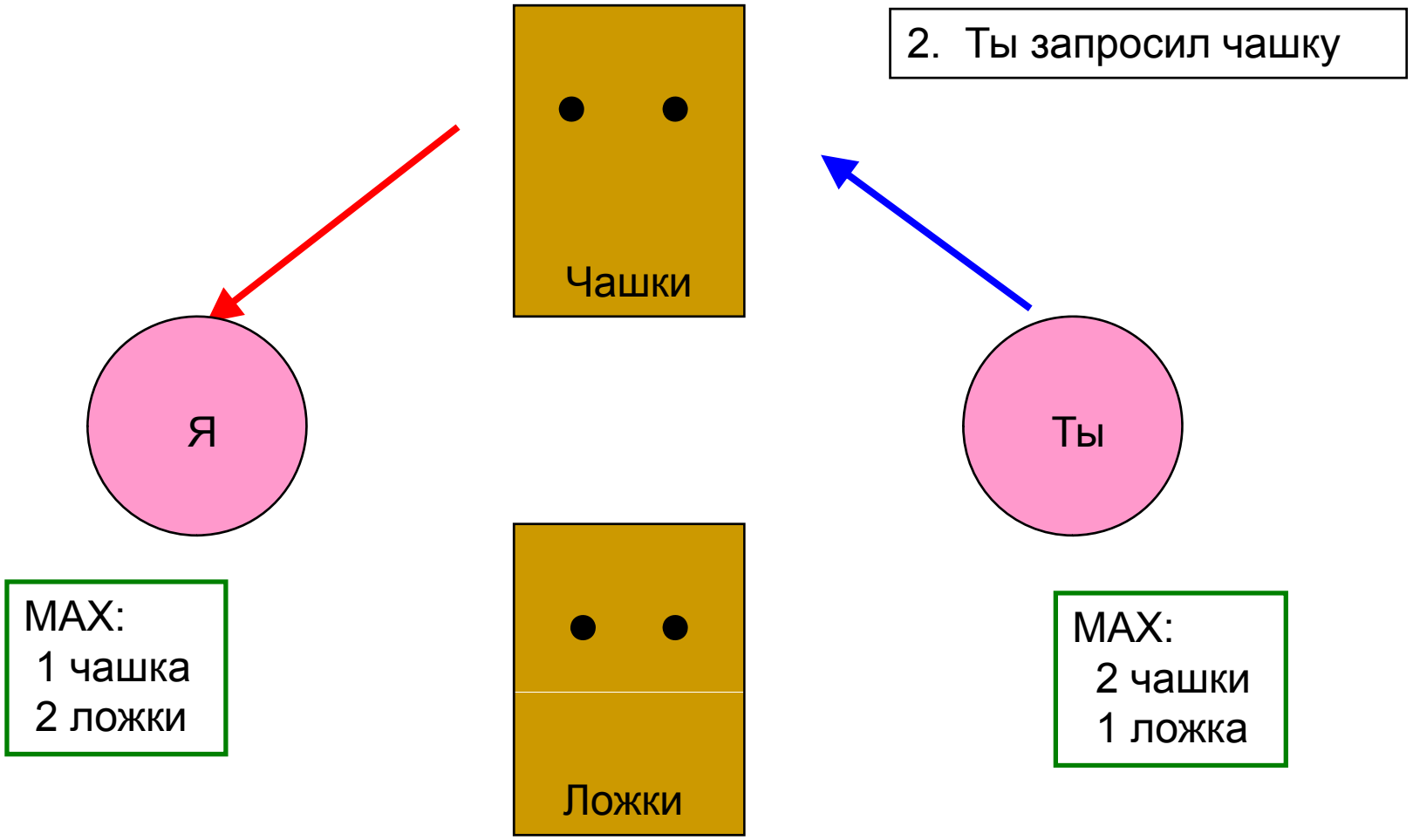


Избегание тупиков...

Запрос удовлетворен – можно сначала полностью выполниться мне, а потом – сможешь выполниться ты

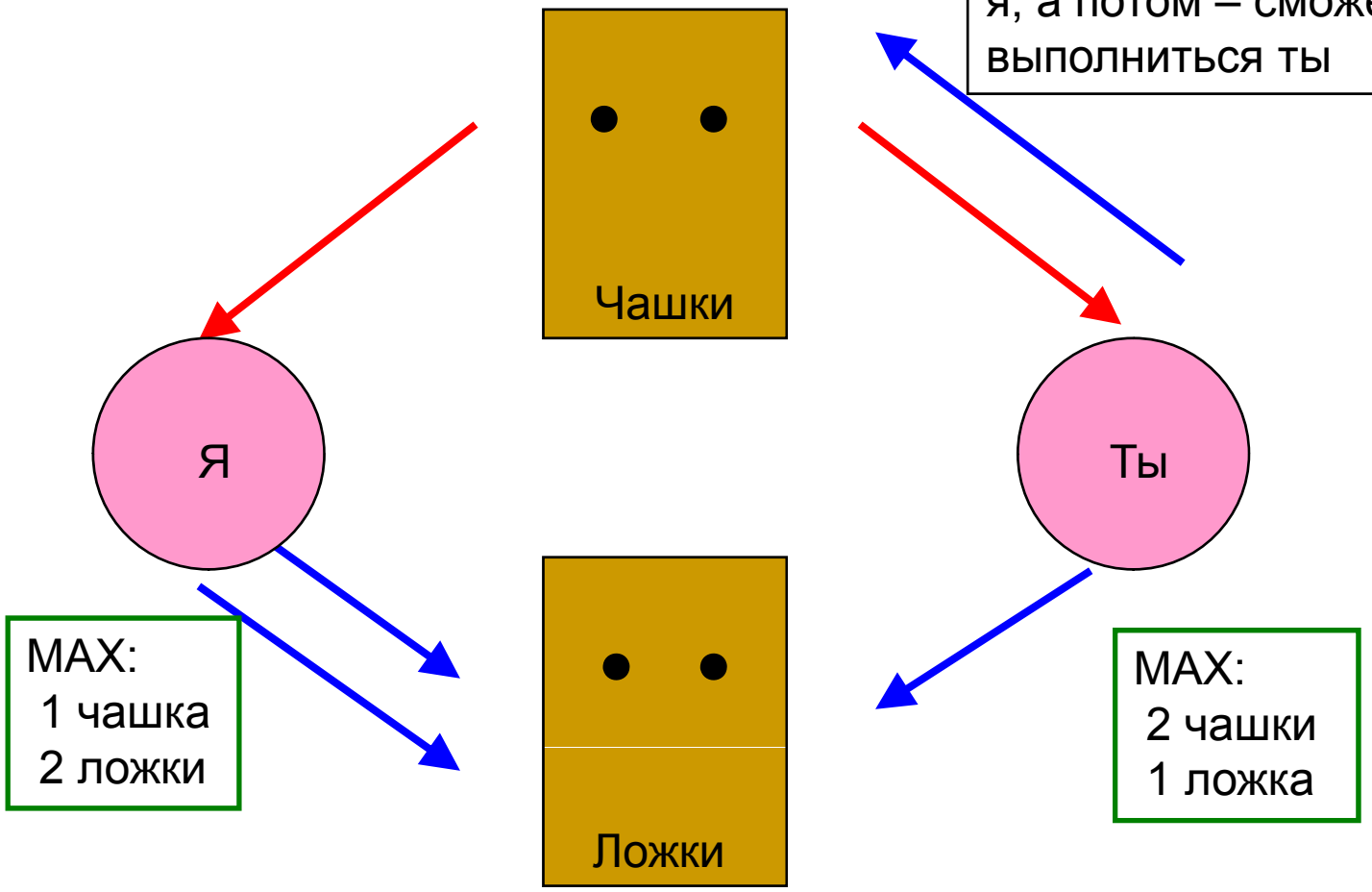


Избегание тупиков…

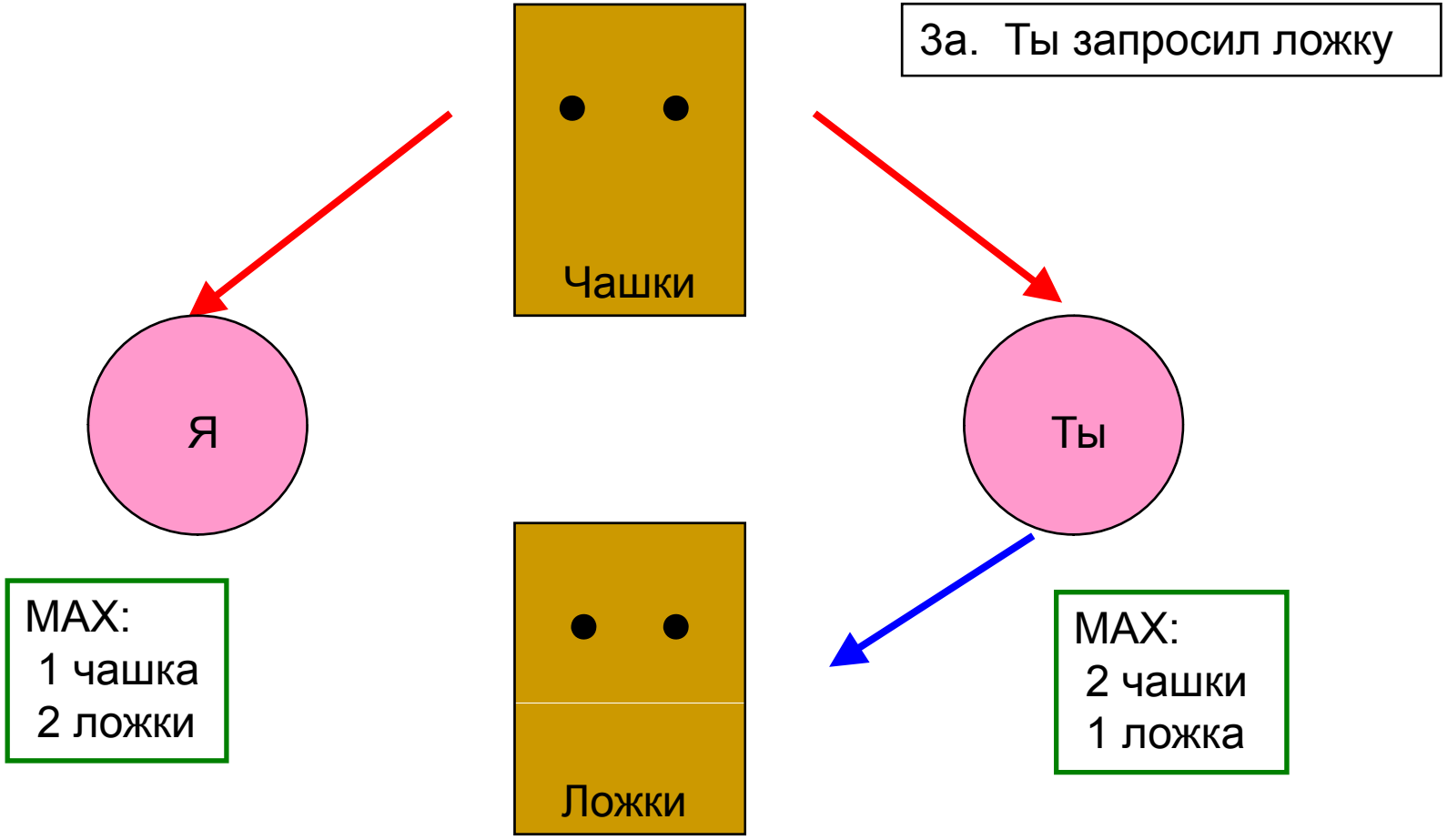


Избегание тупиков...

Запрос удовлетворен – опять могу сначала полностью выполниться я, а потом – сможешь выполниться ты

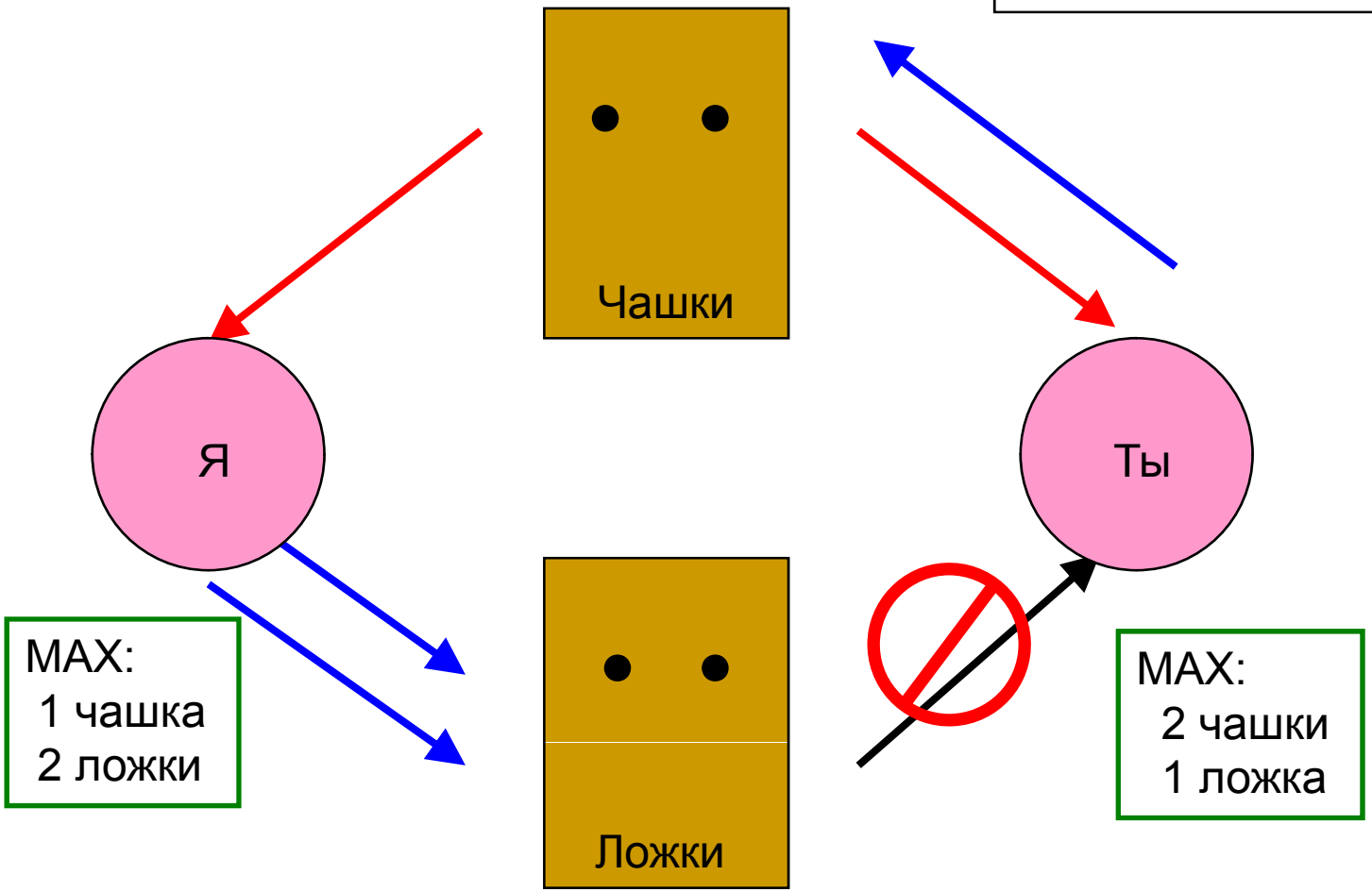


Избегание тупиков…

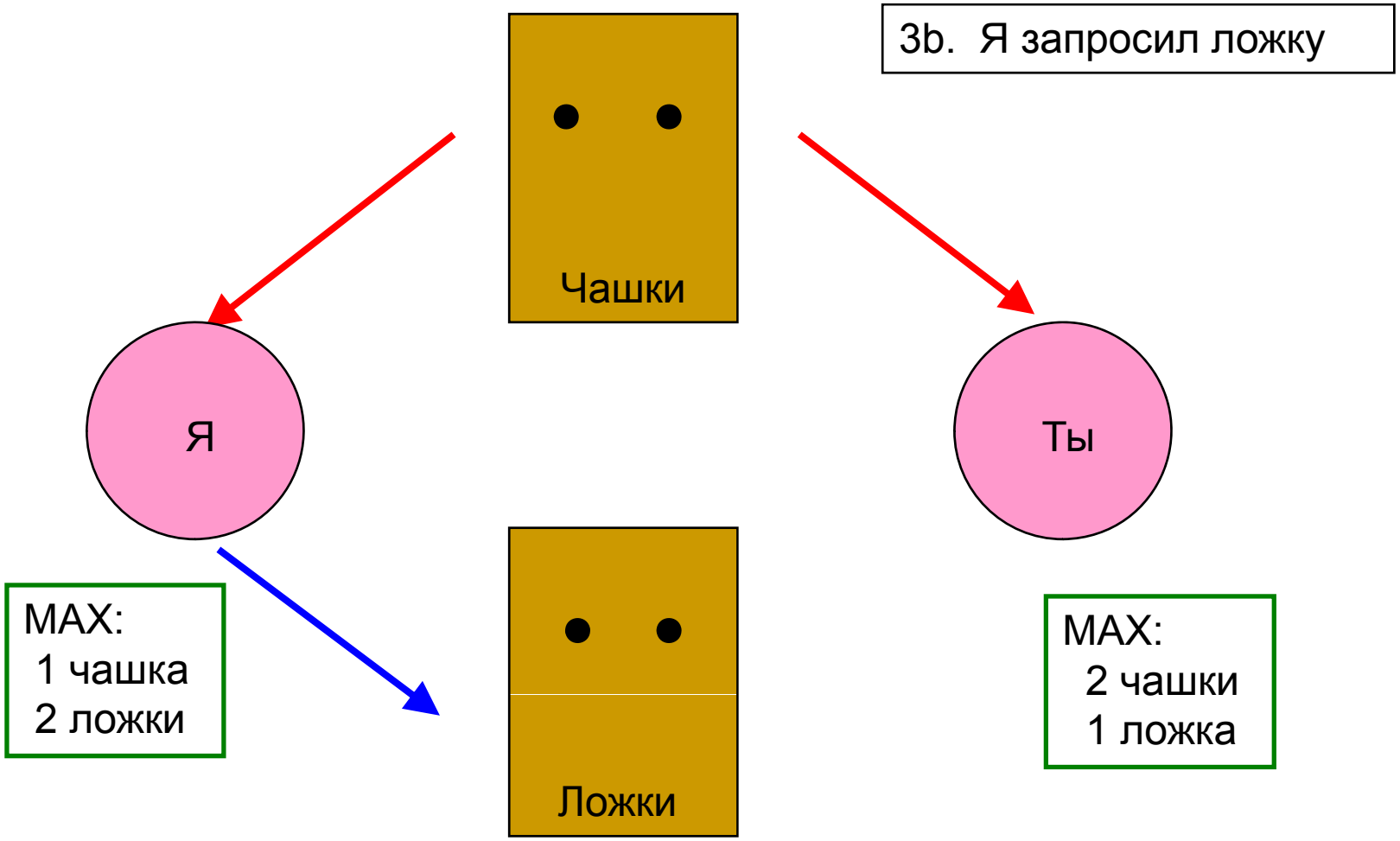


Избегание тупиков...

Нет! Может случиться так, что мы оба не сможем выполниться!

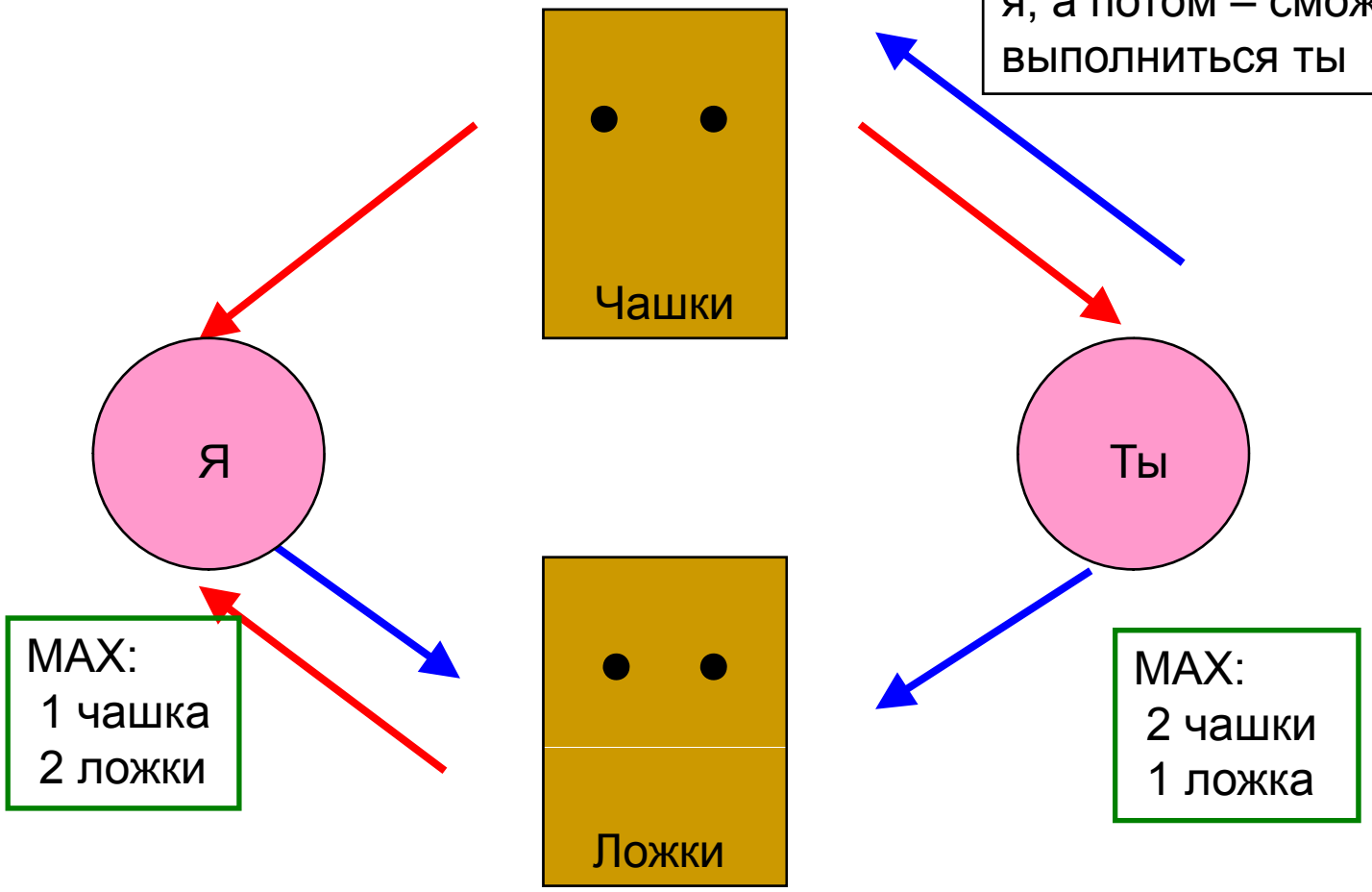


Избегание тупиков…



Избегание тупиков...

Запрос удовлетворен – опять могу сначала полностью выполниться я, а потом – сможешь выполниться ты



Избегание тупиков…

- A,B,C - ресурсы

	Выделено			Max			Доступно		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

- Безопасно ли текущее состояние?

Избегание тупиков

- A,B,C - ресурсы

	Выделено			Max			Доступно		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

- Можно ли удовлетворить запрос P0 на (1,2,2)?

Восстановление после тупика

Обнаружение тупика. Восстановление после тупика

- Если ни один из перечисленных подходов не используется, может возникнуть тупик. В таком случае необходимо:
  - Обнаружить возникновение тупика. Для этого нужно:
    - отслеживать выделение ресурсов (какой процесс каким ресурсом владеет)
    - отслеживать поступающие запросы (какой процесс какой ресурс ожидает)
  - Иметь решения для восстановления из тупика
- Очень накладно поддерживать и обнаружение, и восстановление

Обнаружение тупика

- Когда запускать алгоритм обнаружения тупика?
  - При каждом запросе на выделение ресурса?
  - При каждом запросе, который не может быть немедленно удовлетворен?
  - Каждый час?
  - Когда средняя загрузка ЦП станет меньше 40%?

Восстановление после тупика

- Уничтожение одного/всех процессов, участвующих в тупике
  - Можно продолжать уничтожать процессы, пока тупик не распадется
  - Все вычисления повторяются уничтоженных процессов придется повторить
  - Грубо, но эффективно
- Перераспределение ресурсов между процессами вплоть до разрушения тупика
  - Ресурсы отбираются у владельцев и отдаются другим процессам
- Откат выбранного процесса к некоторой контрольной точке или к началу (partial or total rollback)

Восстановление после тупика Пример – SQL Server

- Запускает алгоритм обнаружения тупика
  - Периодически или
  - По запросу
- Выполняет восстановление посредством
  - уничтожения наименее "дорогого" процесса
  - уничтожения процесса, выбранного в соответствии с приоритетом, указанным пользователем

- "Transaction (Process ID xxx) was deadlocked on (zzz) resources with another process and has been chosen as the deadlock victim. Rerun the transaction."

Заключение

- **Взаимоблокировка (тупик)** – проблема, возникающая при совместном использовании неразделяемых ресурсов
- Существующие средства борьбы с взаимоблокировками
  - "Страусовый" алгоритм
  - Предупреждение взаимоблокировок
  - Избегание взаимоблокировок
  - Обнаружение взаимоблокировок и восстановление после них

Вопросы для обсуждения

- В основе каких "эмпирических" правил программирования лежат принципы безтупикового программирования?
- Может ли система находиться в состоянии, не являющимся ни состоянием взаимоблокировки, ни безопасным состоянием?
- Пусть в системе N процессов и M единиц ресурса, каждому процессу нужно максимум K единиц ресурса. Какое соотношение между N, M и K должно выполняться, чтобы в системе не мог возникнуть тупик?

Задание для самостоятельной работы

- Напишите программу, реализующую алгоритм банкира

Литература

1. Таненбаум Э. Современные операционные системы. 2-е изд. - СПб.: Питер, 2002.
2. Карпов В.Е., Коньков К.А. Введение в операционные системы. Курс лекций. 2-е изд. - М.: ИНТУИТ.РУ, 2005.

Тема следующего раздела – Управление памятью

- Понятие виртуального адресного пространства процесса и его физическая организация
- Управление физической памятью
- Управление содержимым виртуального адресного пространства процесса