Синхронизация в разпределени системи

Проблеми със синхронизацията

- Как процесите се синхронизират в разпределените системи
  - В еднопроцесорни системи, проблемите се решават с използване на методи като например, семафори.
  - Тези методи се базират на споделена памет.
- Как да се определи относителното подреждане на събитията
  - Кое събитие предхожда друго?
  - Трудности при определяне настъпването на събитие в различни машини.

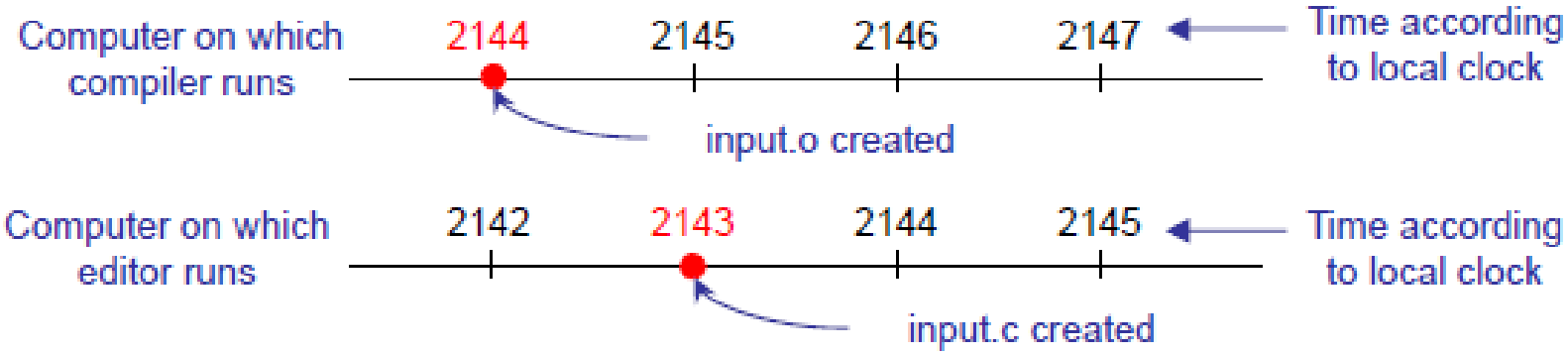
Синхронизация на часовници

- При централизирани системи
  - Времето е ясно как се определя. Процес получава стойността на времето чрез системно извикване. Ако процес А получи времето по-късно и процес В също го получи, то стойността за В ще е по-голяма или равна на стойността за А.
  - Пример: компилация с ***make*** под Linux.

input.c → input.o

Синхронизация на часовници (2)

- При разпределени системи
  - Определянето на времето не е тривиално.



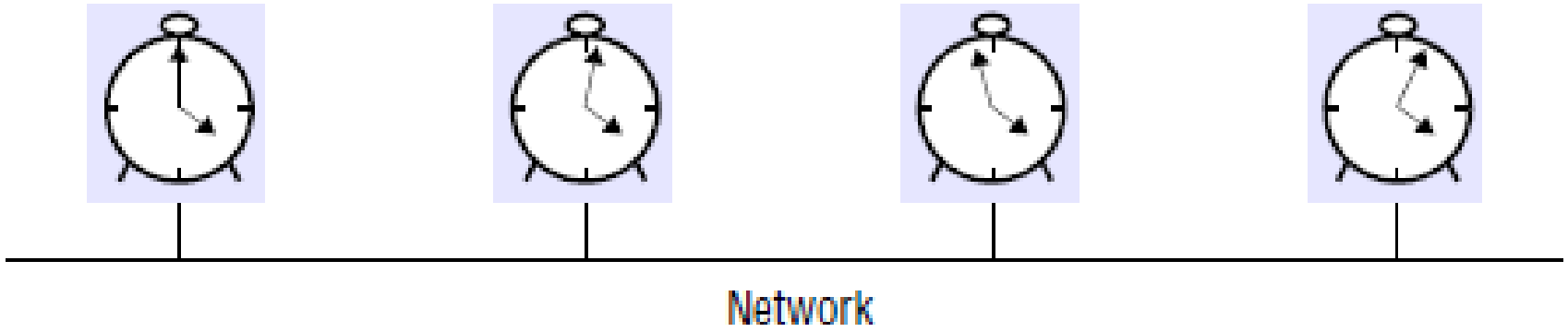
➢ Възможно ли е синхронизирането на всички часовници в разпределена система?

Часовници

- Компютърът има ***таймер***, не часовник.
- Таймерът им кварцов осцилатор.
- Таймерът има два регистъра – броячен и начален:
  - В началният регистър се записва определена стойност.
  - Броячният регистър намаляващ с 1 при всяка осцилация. При стойност 0 се генерира прекъсване (*clock tick*) и се зарежда съдържанието на началния регистър.
  - На всеки clock tick се добавя 1 към времето, което се съхранява в паметта.

Часовници (2)

- При налични *n* компютъра, техните кристали ще имат малки разлики в честотите, което води до постепенно разсинхронизиране (*clock skew*).



Логически и физически часовници

- Синхронизацията на часовниците не е необходимо да бъде абсолютна (Лампорт, 1978):
  - Ако два процеса не си комуникират, техните часовници не е необходимо да бъдат синхронизирани.
  - Необходимо ли е процесите да се базират на това, колко е точното време или по-скоро на **последователността в която се случват събитията**?

Логически и физически часовници (2)

- За алгоритми, при които единствено вътрешната консистенция на часовниците има значение – **логически часовници**.
- За алгоритми, при които часовниците трябва не само да са еднакви, но и да не се отклоняват от реалното време – **физически часовници.**

Физически часовници

- От откриването на механичния часовник през 17 век, времето се измерва астрономически (*mean solar day, mean solar second*)

Mean solar day – средния период на завъртане на Земята

1 секунда = 1 /86 400 от **mean solar day**

Физически часовници (2)

- С откриването на атомния часовник през 1948г, времето може вече да се измерва по-прецизно
  - По света има лаборатории с атомни часовници, които периодично дават информация на Bureau International de l'Heure (BIH) в Париж колко пъти техния часовник е издал clock tick.
  - BIH усреднява тези стойности за формиране на Temps Atomique Internacional (TAI)
  - Оригинално, атомното време се изчислява, така, че да се направи **атомната секунда** равна на **mean solar second**.

Универсално координирано време

- Поради факта, че **mean solar day** става по дълго, BIH извърша корекция (чрез прескачане на секунда) – Universal Coordinated Time (UTC)
- UTC се предоставя от:
  - National Institute of Standard Time (NIST) чрез късовълнова станция с позивна „WWV” от форт Колинс, Колорадо: точност ± 1 msec.
  - От всяка наземна GEOS станция: точност до 0.5 msec
  - През телефон от NIST: евтино, но неточно.

Синхронизиране на физически часовници

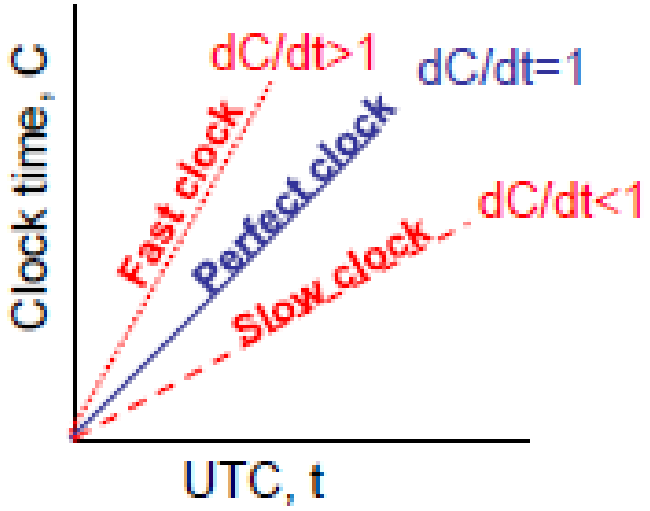
- Модел на системата:
  - Таймерът на всяка машина генерира прекъсване *H* пъти в секунда (теоретично).
  - При изтичане на таймера, обработващата прекъсването програма добавя 1 към софтуерния часовник.
  - Ако UTC е *t*, то стойността на часовника на машината *p* е $C_P(t)$.
  - В идеалния случай, $C_P(t) = t$ за всички *p* и всички *t*, т.е. $dC/dt = 1$.

Синхронизиране на физически часовници

- Модел на системата:
  - На практика, таймерите не генерират прекъсвания точно *H* пъти в секунда. Относителната грешка на съвременните чипове е 10^{-6} – 1 сек / 1000000 сек -> 11.6 дена.
  - Всеки таймер си спецификация на функциониране.
  - $1 - \rho \leq dC/dt \leq 1 + \rho$, където константата ρ е максималната скорост на отклонение (указана от производителя)

Синхронизиране на физически часовници (2)

- Ако след синхронизацията, два часовника се отклоняват от UTC в противоположни посоки, те могат да бъдат **2ρΔt** един от друг за време **Δt** .
- За часовници, които не се различават повече от **δ**, те трябва да се ресинхронизират всеки **δ / 2ρ** секунди.



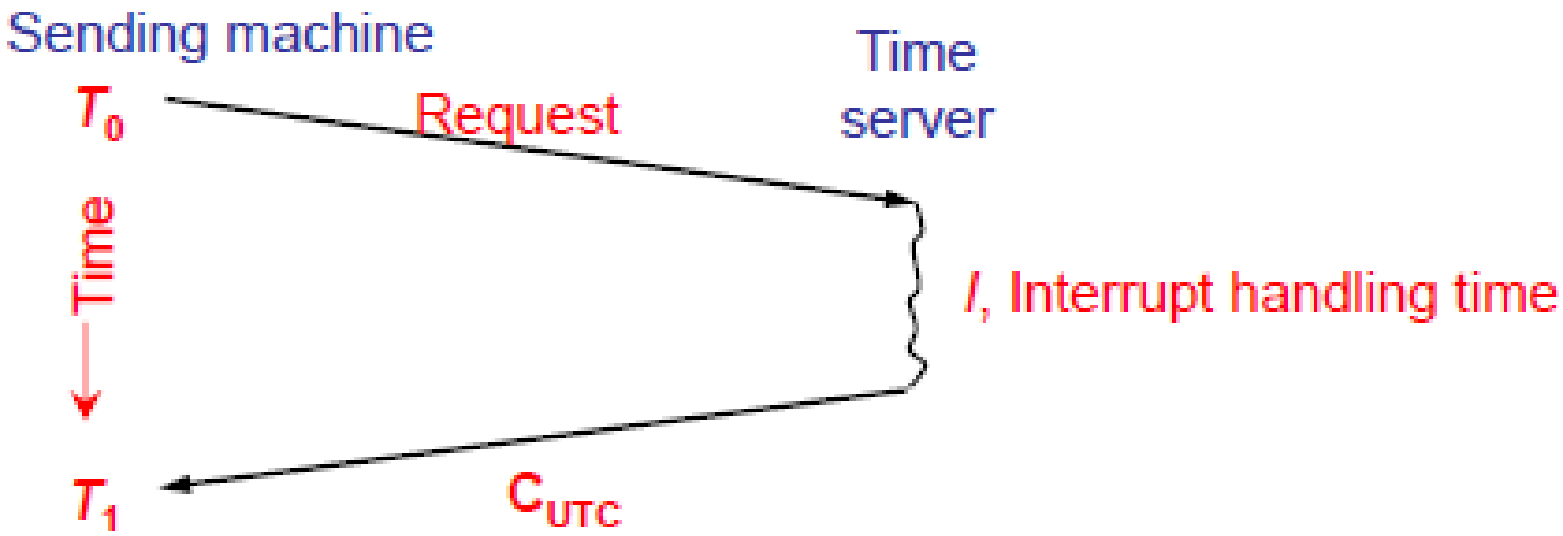
Алгоритъм на Cristian

- Използва се времеви сървър с WWV приемник. Останалите машини се синхронизират с него.
- Времето T_1 не може да се установи на C_{UTC} тъй като времето никога не тече обратно. Промените се извършват постепенно с добавяне на секунди за всяко прекъсване.
- Времето за предаване се включва в промените.

Алгоритъм на Cristian (2)

- Времето за предаване се изчислява като

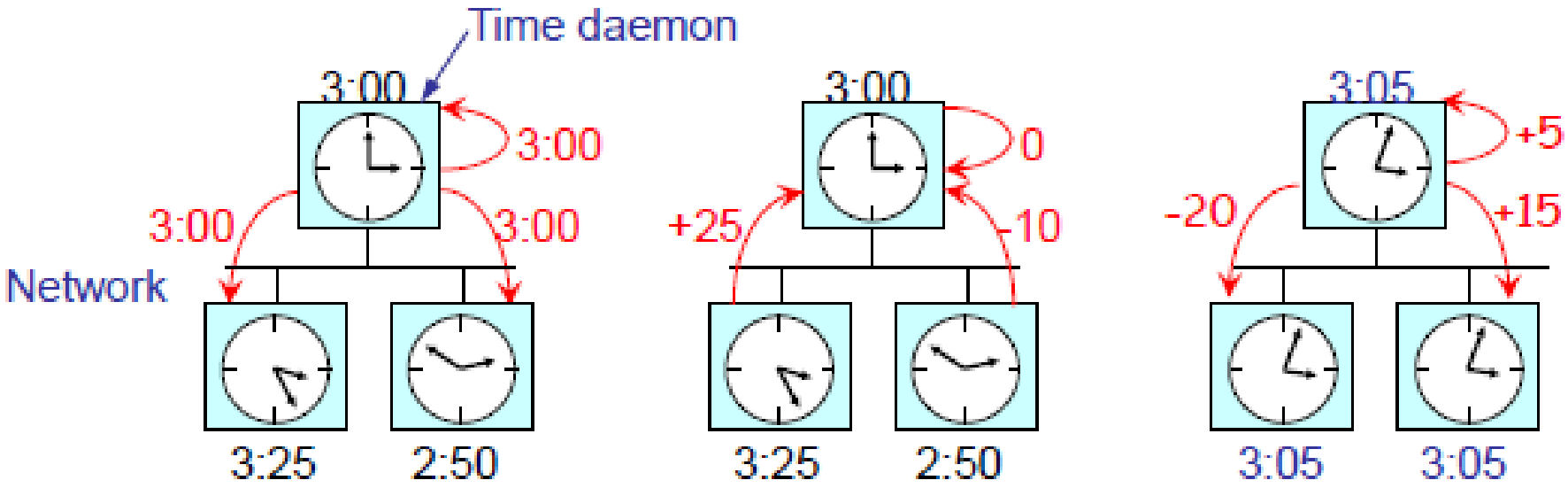
$$(T_1 - T_0 - I) / 2$$



Алгоритъм на Berkeley

- Подходящ в случаите когато не е налице машина с WWV приемник.
- Използва се времеви сървър (time daemon), който е активен компонент.
  - ВС периодично пита всяка машина за нейното време.
  - На базата на отговорите изчислява средното време.
  - Изпраща на машините указания да ускорят или забавят своите часовници.
- Времевият сървър се настройва периодично ръчно от администратора.

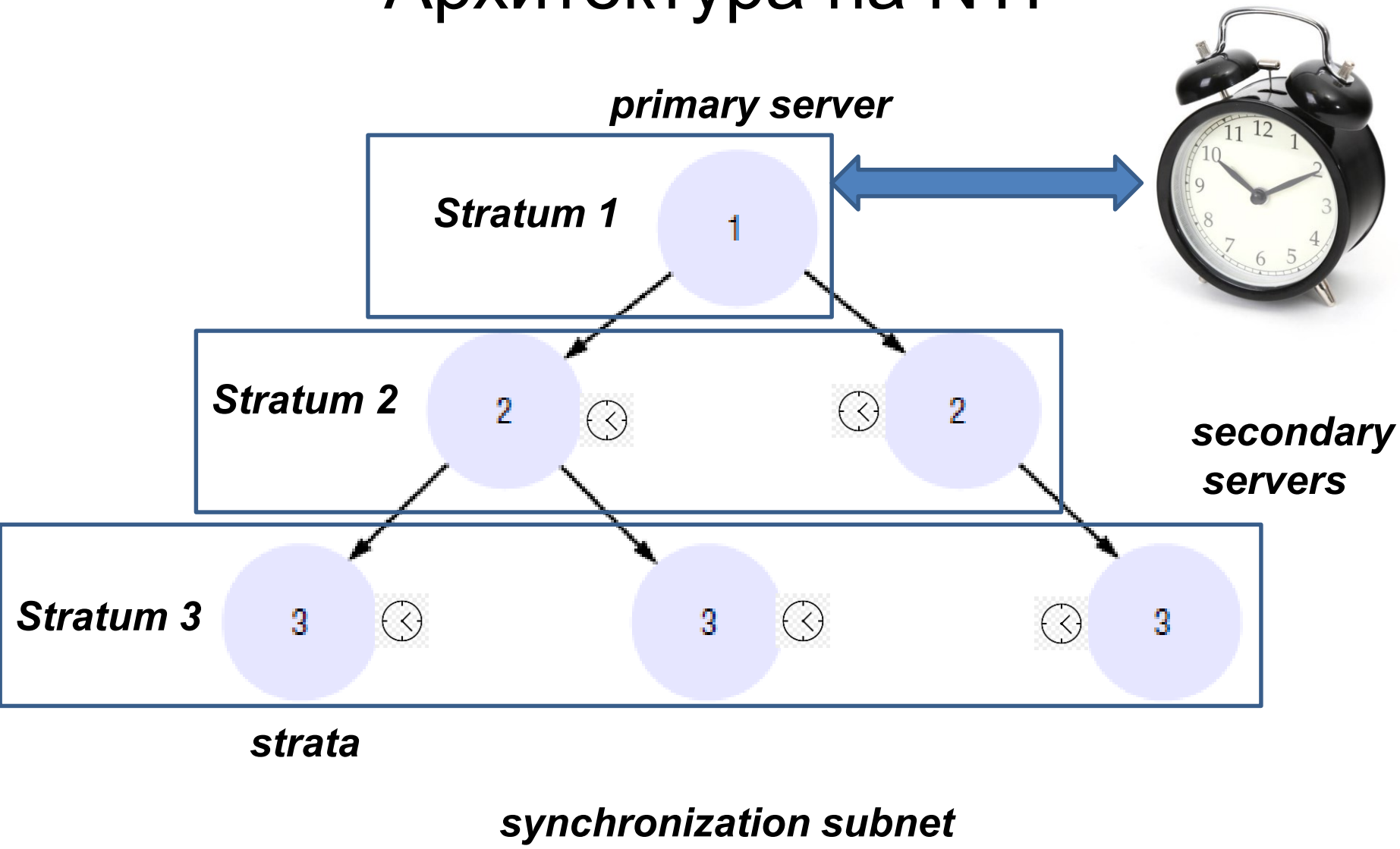
Алгоритъм на Berkeley (2)



Internet Network Time Protocol

- Предоставя услуга за синхронизиране на клиентите през Интернет към UTC.
- Предоставя надеждна услуга чрез множество резервирани сървъри и връзки.
- Поддържа на нарастващ брой клиенти и сървъри.
- Осигурява защита срещу опити за манипулиране на информацията.

Архитектура на NTP



Функциониране на NTP

- Сървърите се синхронизират в три режима:
  - *Multicast* – един сървър периодично разпространява времето към останалите. Те установяват своето време, допускайки малка времезадръжка. Удобен при високоскоростни LAN. Не толкова точен.

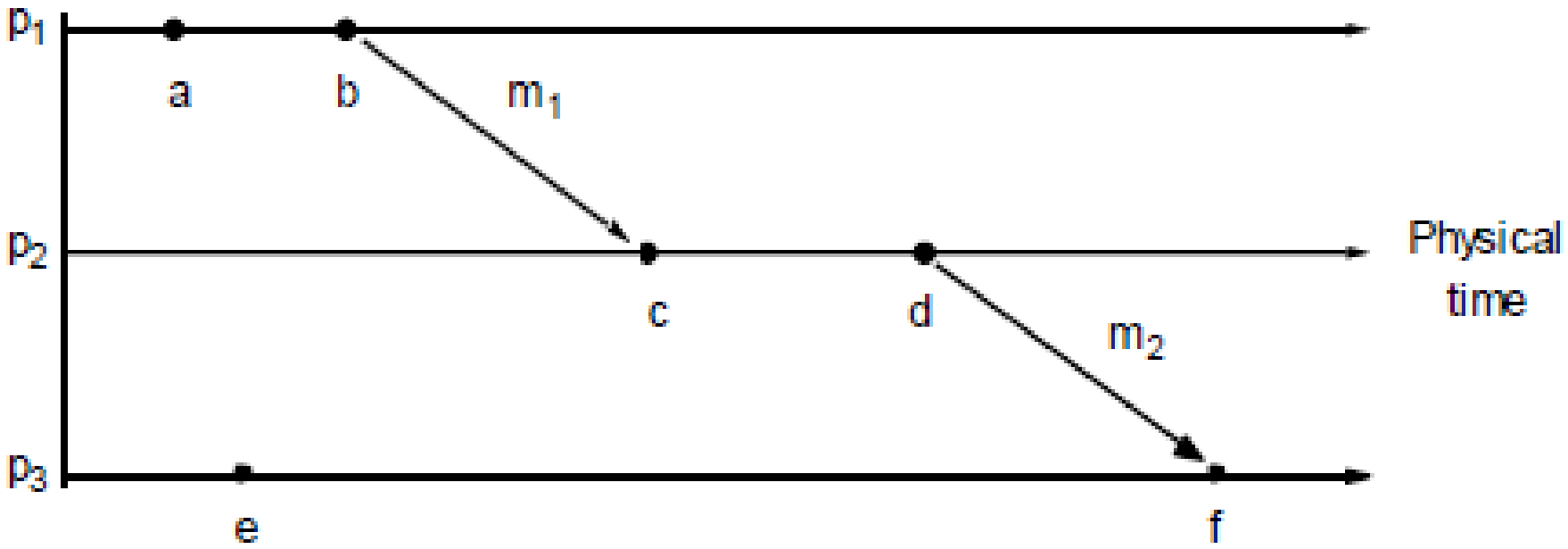
Функциониране на NTP (2)

- *Procedure-call* – един сървър периодично получава заявки от други машини, като отговаря със стойността на текущия си часовник. Подходящ за висока точност.
- *Symmetric* – двойка сървъри си обменят информация за времето. Позволява точно синхронизиране.

Синхронизиране на логически часовници

- Отношение *happens-before*
  - Ако ***a*** и ***b*** за събития в един процес и ***a*** настъпи преди ***b*,** то $a \rightarrow b$ е истина.
  - Ако **a** е събитието на изпращане на съобщение от един процес, и **b** е събитието на получаване на същото съобщение от друг процес, то $a \rightarrow b$ също е истина.
  - Ако две събития **a** и **b** настъпват в различни процеси, които не обменят съобщения, то нито $a \rightarrow b$ е истина, нито пък $b \rightarrow a$ е истина. Тези събития се дефинират като „конкурентни“ ($a \parallel b$).
  - Happens-before е транзитивно отношение: $a \rightarrow b$ и $b \rightarrow c$ следва $a \rightarrow c$

Синхронизиране на логически часовници (2)



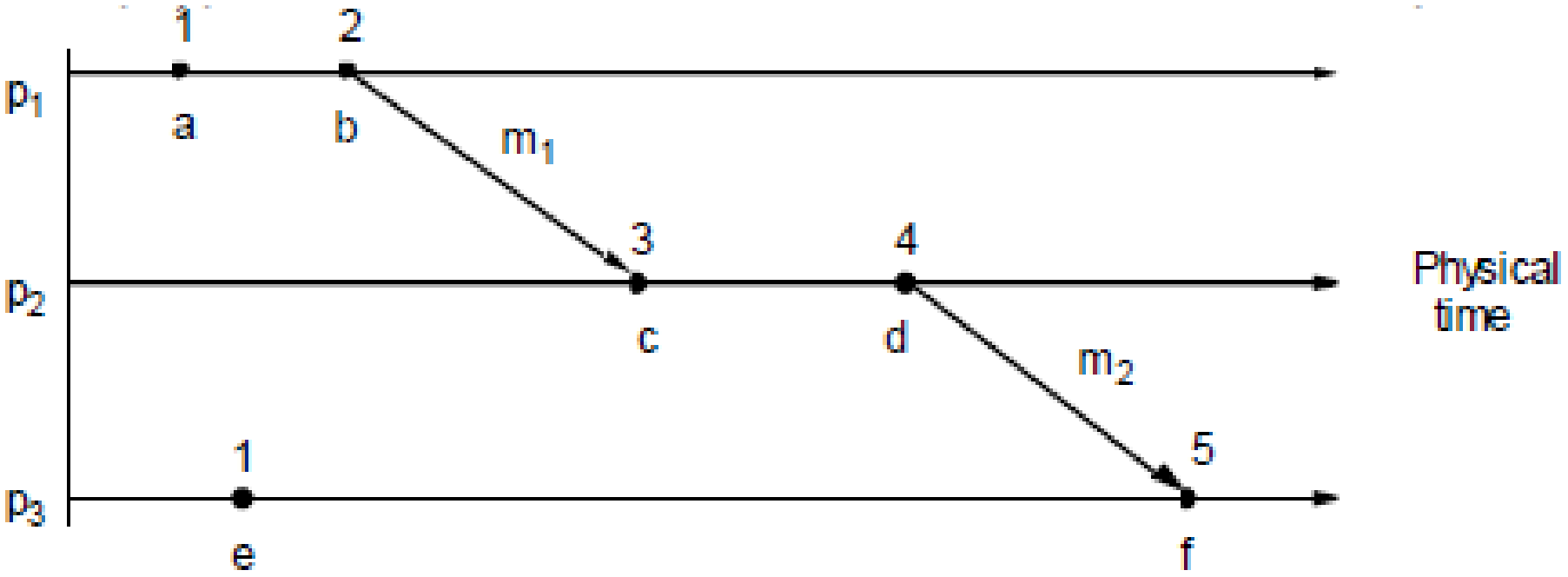
Алгоритъм на Лампорт

- Базира се на отношение *happens-before*.
- Всеки процес p_i има локално монотонно нарастващ брояч – логически часовник L_i.
- На всяко събитие *e*, което настъпва в процеса p_i се присвоява времеви маркер $L_i(e)$.

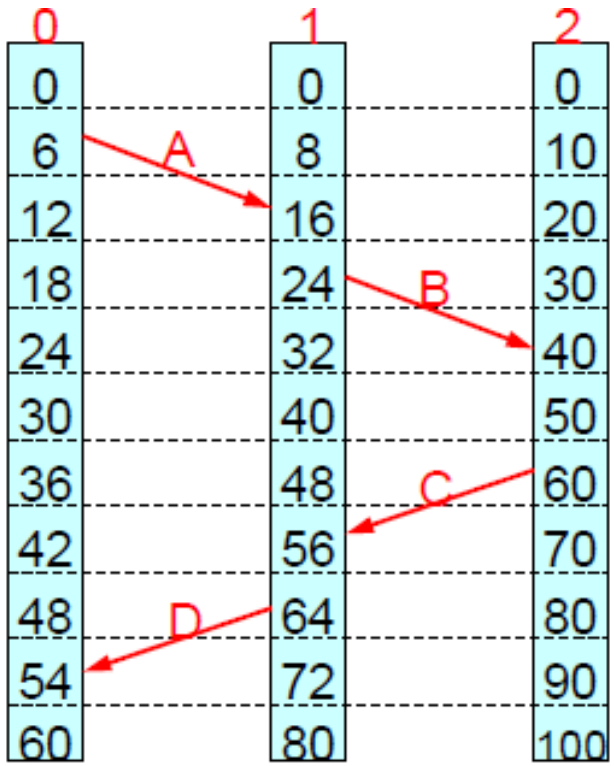
Алгоритъм на Лампорт (2)

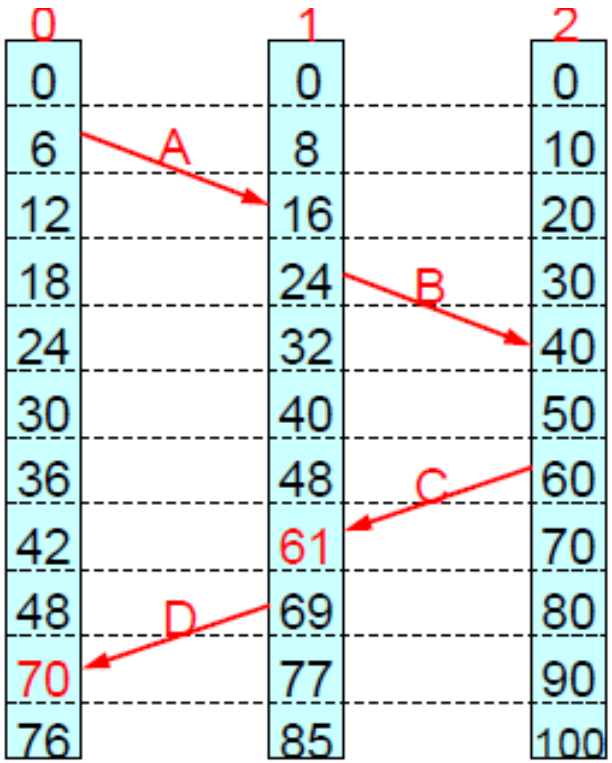
- Правила:
  - L_i се увеличава преди да настъпи събитието *e* в процеса p_i : $L_i = L_i + 1$.
  - Когато p_i изпраща съобщение *m*, той добавя времеви маркер $t = L_i$: (m, t). Това е събитието *send(m)*.
  - При получаване на (m, t), процесът p_i изчислява неговия логически часовник като $L_j := max(L_j , t)$; $L_j := L_j + 1$; времето на събитието *receive(m)*.

Алгоритъм на Лампорт (3)



Пример


0
1
2
0 6 12 18 24 30 36 42 48 54 60
0 8 16 24 32 40 48 56 64 72 80
0 10 20 30 40 50 60 70 80 90 100
A
B
C
D


0
1
2
0 6 12 18 24 30 36 42 48 70 76
0 8 16 24 32 40 48 61 69 77 85
0 10 20 30 40 50 60 70 80 90 100
A
B
C
D

Векторни времеви маркери

- Времеви маркер на Лампорт
  - Ако $e \rightarrow f$, то *timestamp(e)* < *timestamp(f)*.
  - Но ако *timestamp(e)* < *timestamp(f)* това не означава, че $e \rightarrow f$.
  - Не обхваща причинно-следствената връзка (*causality*).
- Векторен времеви маркер
  - Всички машини използват вектор от логически часовници, където *i*-ят елемент е стойността на часовника на *i*-тата машина.
  - Елементите на вектора за другите машини в началото са със стойност 0.

Векторни времеви маркери (2)

- За всяко не-комуникационно събитие в машина *i*:
  - Увеличаване с 1 на *i*-я елемент във вектора.
- При изпращане на съобщение се изпраща целия вектор.
- При получаване на съобщение от машина *j* (на машина *i*), векторът се обновява като:
  - Увеличаване на на *i*-я елемент.
  - За останалите елементи, взема се максималната стойност между векторите *i* и *j*.

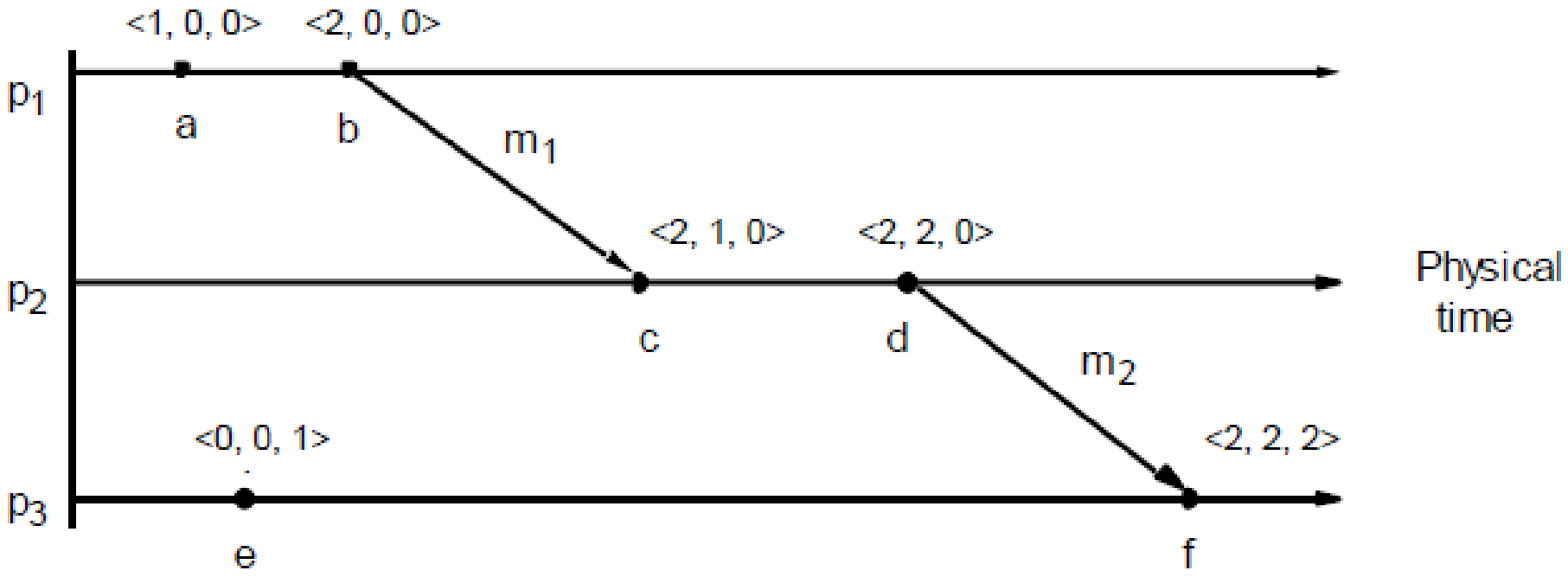
Векторни времеви маркери (3)

- Два времеви маркера са равни, ако и само ако всички елементи са равни:

$$VT1[i] = VT2[i],\ i = 1, ..., n$$

- VT1 <= VT2
  - Ако и само ако VT1[i] <= VT2[i], I = 1, …,n
- VT1 < VT2
  - Ако и само ако VT1 <= VT2 и VT1 != VT2
- В противен случай, VT1 е конкурентен с VT2

Векторни времеви маркери (3)



Избиране на координатор

- Много алгоритми изискват един процес да играе роля на координатор.
- По принцип, няма значение кой процес ще играе тази роля, но поне един трябва да е такъв.
- Как да се направи избор на координатор?

Избиране на координатор – общ подход

- Допускания:
  - Всеки процес има уникален номер, който го идентифицира.
  - На една машина има само един процес.
  - Всеки процес знае номерата на останалите процеси.
  - Процес не знае кои процеси са действащи и кои не се изпълняват.
- Подход:
  - Да се открие процесът с най-висок номер и да се назначи за координатор.
  - Алгоритмите за избор се различават по начините на откриване на местоположението.

Bully алгоритъм

- Ако процес P открие, че координаторът не отговаря на заявките, инициира нов избор:
  - P изпраща съобщение ELECTION към всички процеси с по-големи номера.
  - Ако няма отговор, P печели и става координатор.
  - Ако отговори някой с по-висок номер, P приключва участието си в избора.

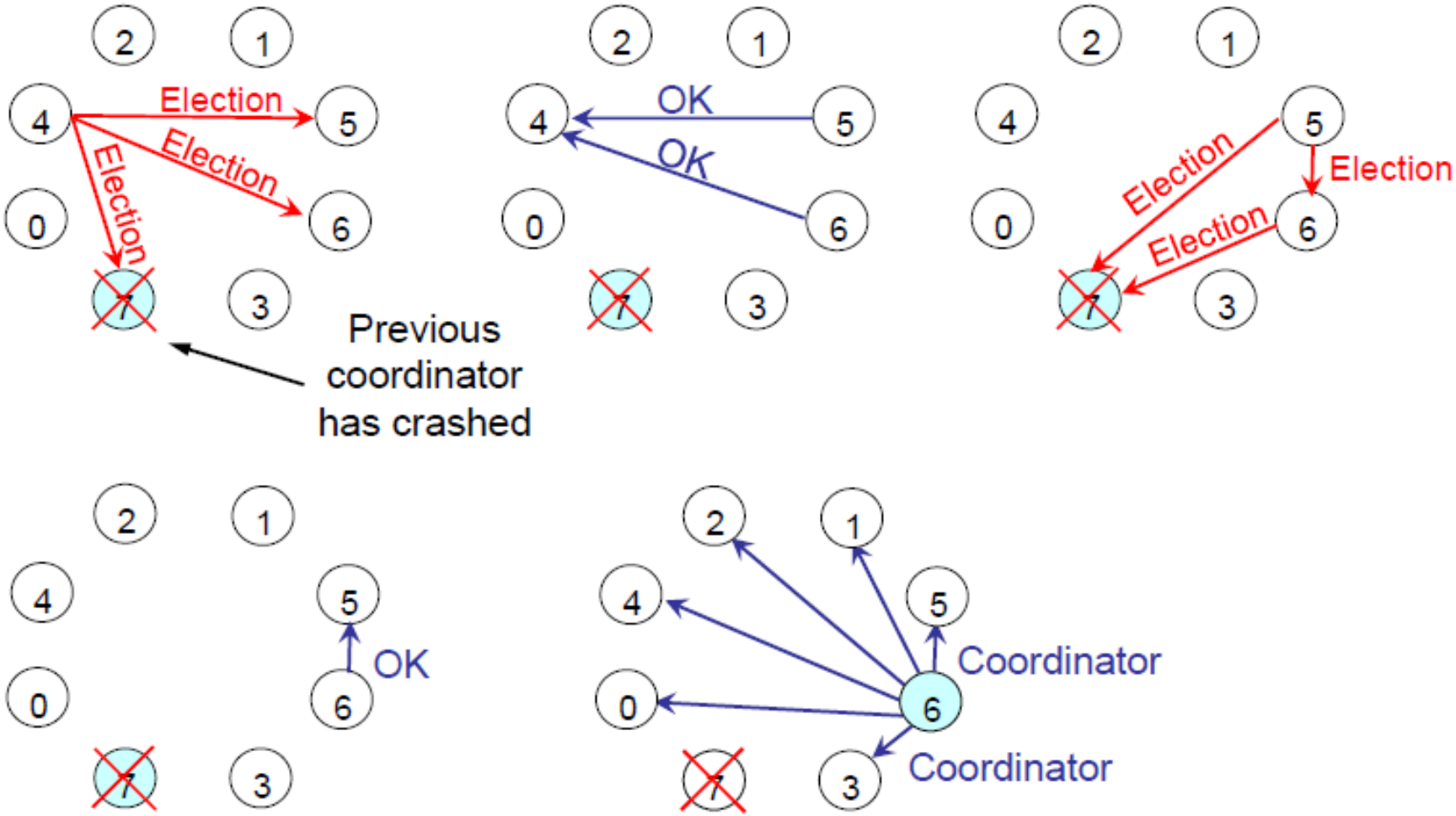
Bully алгоритъм (2)

- Когато процес получи съобщението ELECTION от процеси с по-малки номера:
  - Получателят изпраща съобщение OK обратно на изпращача с което го уведомява, че е действащ и ще продължи избора.
  - Получателят провежда избор, ако вече не е го е приключил.
  - Евентуално, всички процеси ще се откажат, с изключение на един, който ще новия координатор.
  - Новият координатор известява победата си с изпращане на съобщение COORDINATOR към останалите процеси.
- Ако преди това отпаднал процес се възстанови:
  - Провежда избор. Ако той е с най-големия номер от активните процеси, той ще спечели избора.

Bully алгоритъм (3)

- **Най-якото момче винаги печели**

Bully алгоритъм - пример



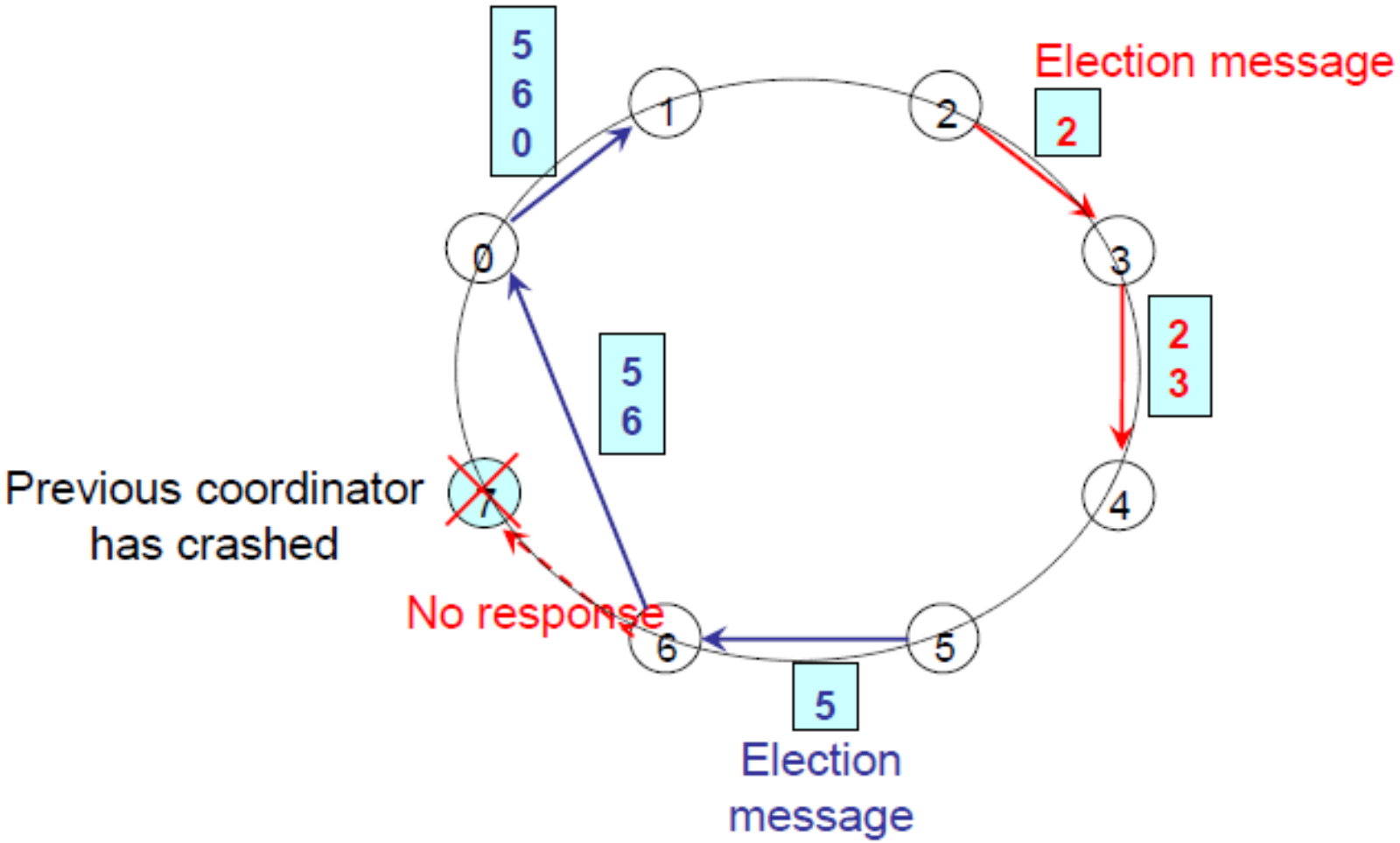
Ring алгоритъм

- Процесите са физически или логически свързани в кръг. Всеки процес знае кой е след него. Не се използват маркери (*tokens*).
- Алгоритъм:
  - Когато процес открие, че координаторът не функционира:
    - Формира съобщение ELECTION с неговия номер на процес.
    - Изпраща съобщението към този след него. Ако той е отпаднал, прескача ги и изпраща към следващия поред в кръга или до следващ, който е активен.
    - На всяка стъпка, изпращачът добавя своя номер на процес в списък в съобщението.

Ring алгоритъм (1)

- Когато съобщението се върне обратно към процеса, който е започнал:
  - Процесът открива съобщението, което съдържа неговия номер
  - Променя типа на съобщението на COORDINATOR.
  - Изпраща отново съобщението да премине през всички за да ги информира за: кой е новия координатор (член от списъка с най-големия номер) и кои са членовете на новия кръг.
  - Когато съобщението се завърне, то се унищожава.

Ring алгоритъм (2)



Взаимно изключване

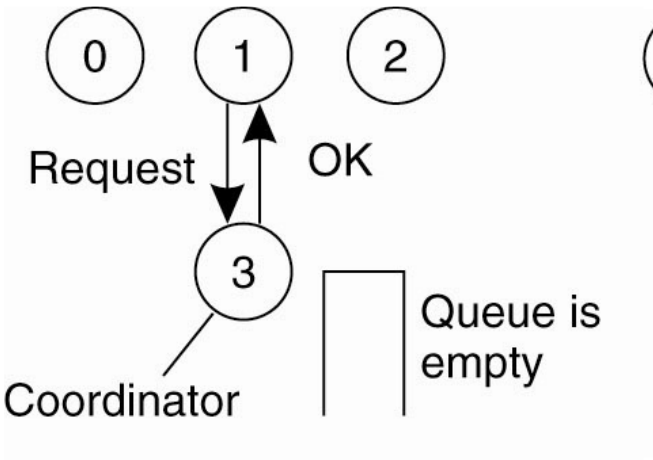
- Характерно за разпределените системи е конкурентността и съвместната работа на множество процеси.
- Аналогично на еднопроцесорните системи, разпределените процеси могат едновременно да осъществят достъп до едни и същи ресурси.
- Необходими са решения за осигуряване на взаимно изключване.

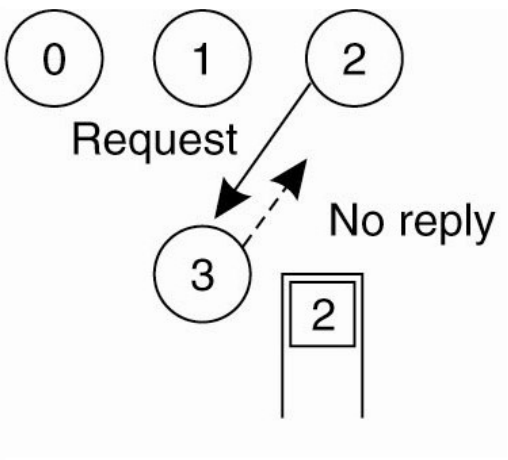
Критична секция

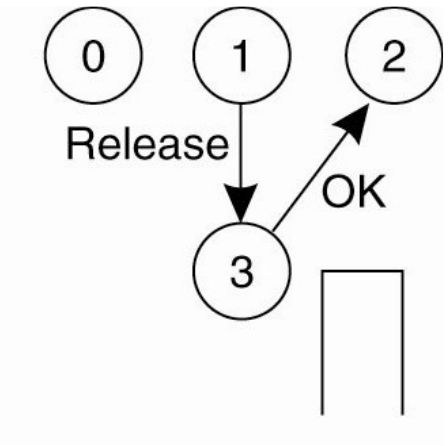
- **Критична секция (Critical Section - CS)** – тази част от кода на нишка, в който има обръщение към общ ресурс.

- Изисквания към CS:
  - В критична секция в даден момент може да се намира само една нишка.
  - Ако една или няколко нишки желаят да влязат в критичните си секции, то на една от тях това задължително трябва да бъде разрешено.
  - Всяка нишка може да преминава в произволно състояние извън критичната си секция, т.е. може да бъде дори спряна. Това не трябва да оказва влияние върху другите нишки при използване на общия ресурс.
  - Относителните скорости на развитие на нишките са предварително неизвестни и произволни.
  - Нишка не може да спре изпълнението си в критична секция.
  - Критичните секции не трябва да имат приоритети.

Решение с централизиран сървър





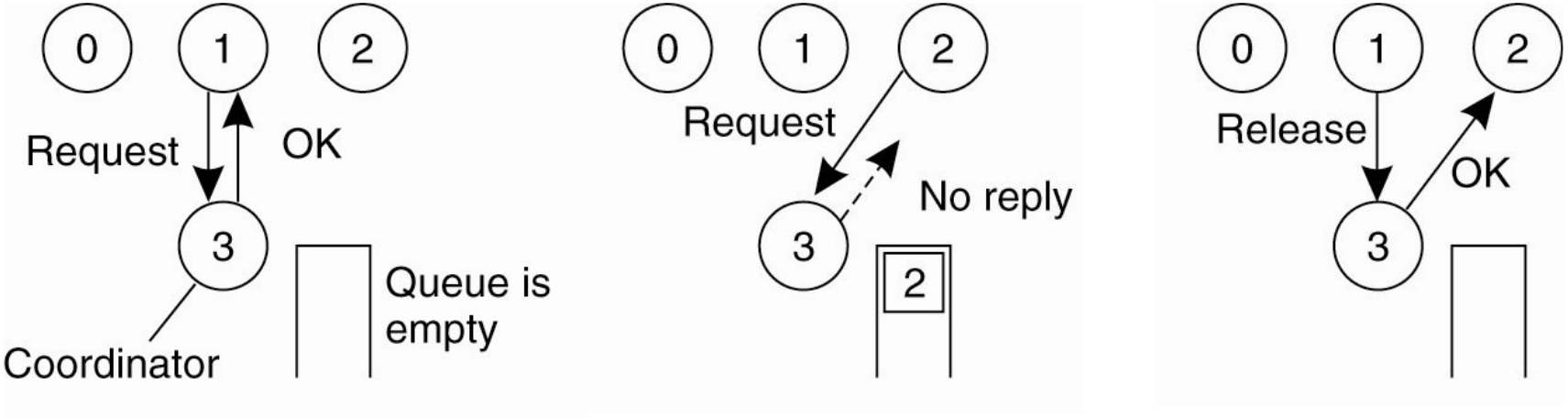


P1 пита координатора за разрешение за достъп. Получава.

P2 пита координатора за разрешение за достъп до същия ресурс. Няма отговор.

Когато P1 освободи ресурса се обажда на координатора. Той отговаря на P2.

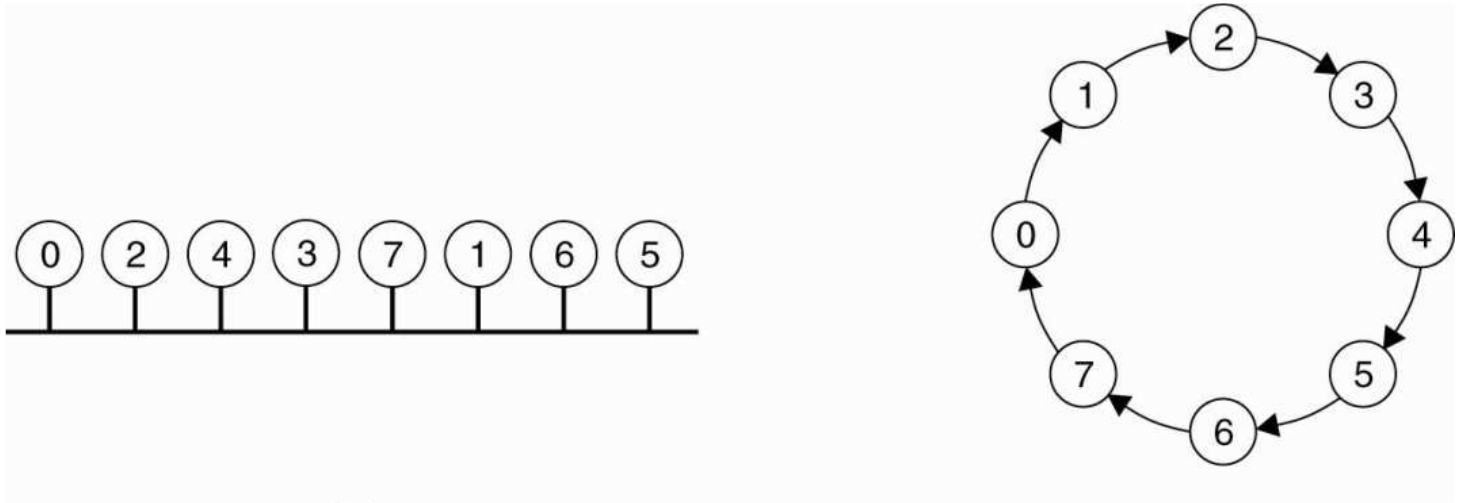
Решение с централизиран сървър - проблеми



- Ако координатора откаже?

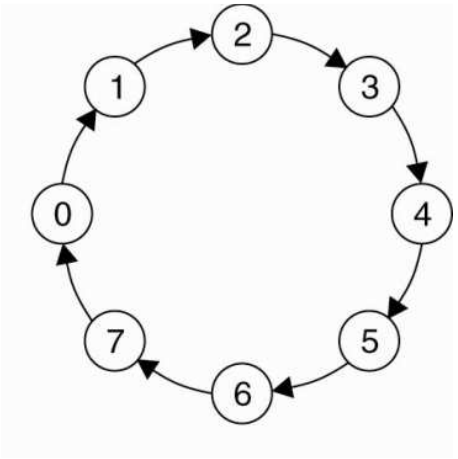
Token ring алгоритъм

- Процесите се организират в логически кръг. Предава се маркер между тях.
- Когато процес получи маркера, може да достъпва общия ресурс.



Token ring алгоритъм

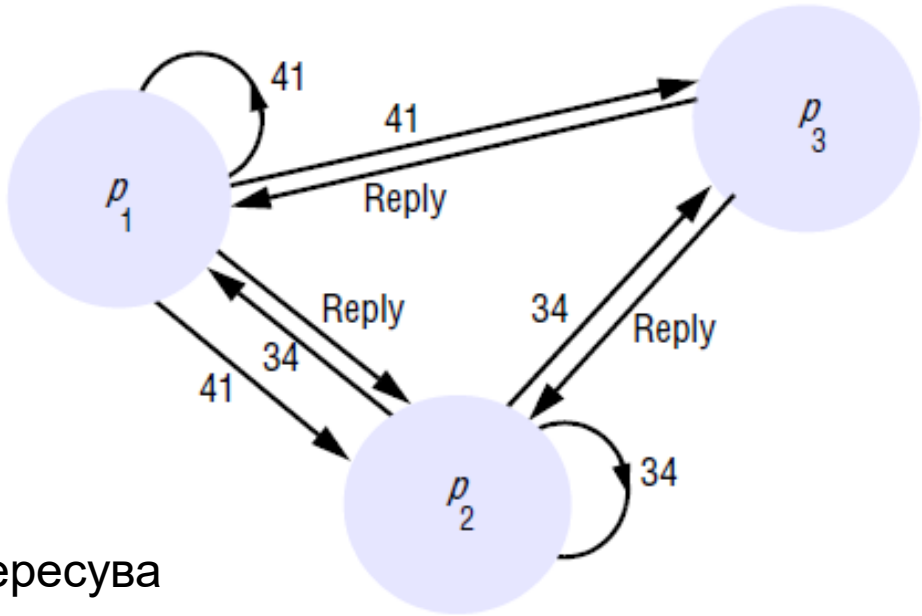
- Процес 0 пуска маркер. Маркерът циркулира.
- Предаването е от k към k+1 в съобщения от типа *point-to-point*.
- Процес получава маркера – може да достъпи критичната си секция, ако е необходимо
- След приключване, предава маркер по кръга. Не може да се използва повторно същия маркер от процес.



Алгоритъм с мултикаст и логически часовници

- Ако процес иска да влезе в критичната си секция, изпраща заявка по мултикаст.
- Ще влезе само ако всички останали му отговорят.
- Използва се синхронизация на Лампорт.
- Отговор се получава ако:
  - Получилият заявката процес няма да влиза в своята КС
  - Има намерение, но има по-малък приоритет (на базата на времевите маркери)
  - В останалите случаи отговорът се отлага.

Алгоритъм с мултикаст и логически часовници



- P3 не се интересува
- P1 и P2 заявяват едновременно
- При получаване на заявка P2 вижда, че тя е с по-голям времеви маркер от неговата и не отговаря
- P1 вижда, че заявката на P2 е с по-малък времеви маркер от неговата и отговаря.
- P2 след получаване на отговора влиза в КС