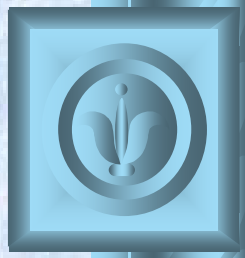


# فصل ششم: همگام سازی

## سیستمهای توزیع شده

امیر مسعود رحمانی

# همگام سازی

۱- همگام سازی ساعت ها

۲- همگام سازی فرایندها

در دستیابی انحصاری: فرآیند ها نباید به طور همزمان و همروند به منبع مشترکی، مثل چاپگر دستیابی داشته باشند.

در بسیاری موارد، مهم است گروهی از فرآیند ها بتوانند برای دستیابی انحصاری به منابع یک فرآیند را به عنوان هماهنگ کننده بپذیرند، که می تواند بر اساس الگوریتم انتخاب، صورت گیرد.

# همگام سازی ساعت (۱)

در سیستم متمرکز زمان مبهم نیست هر پروسس با درخواست زمان میتوان آخرین زمان را دریافت نماید. ولی در سیستم توزیع شده، توافق بر سر زمان، ساده نیست.

مثال) روش کار در برنامه یا یوتیلیتی **make** یونیکس (ترکیب فایلهای یک پروژه)

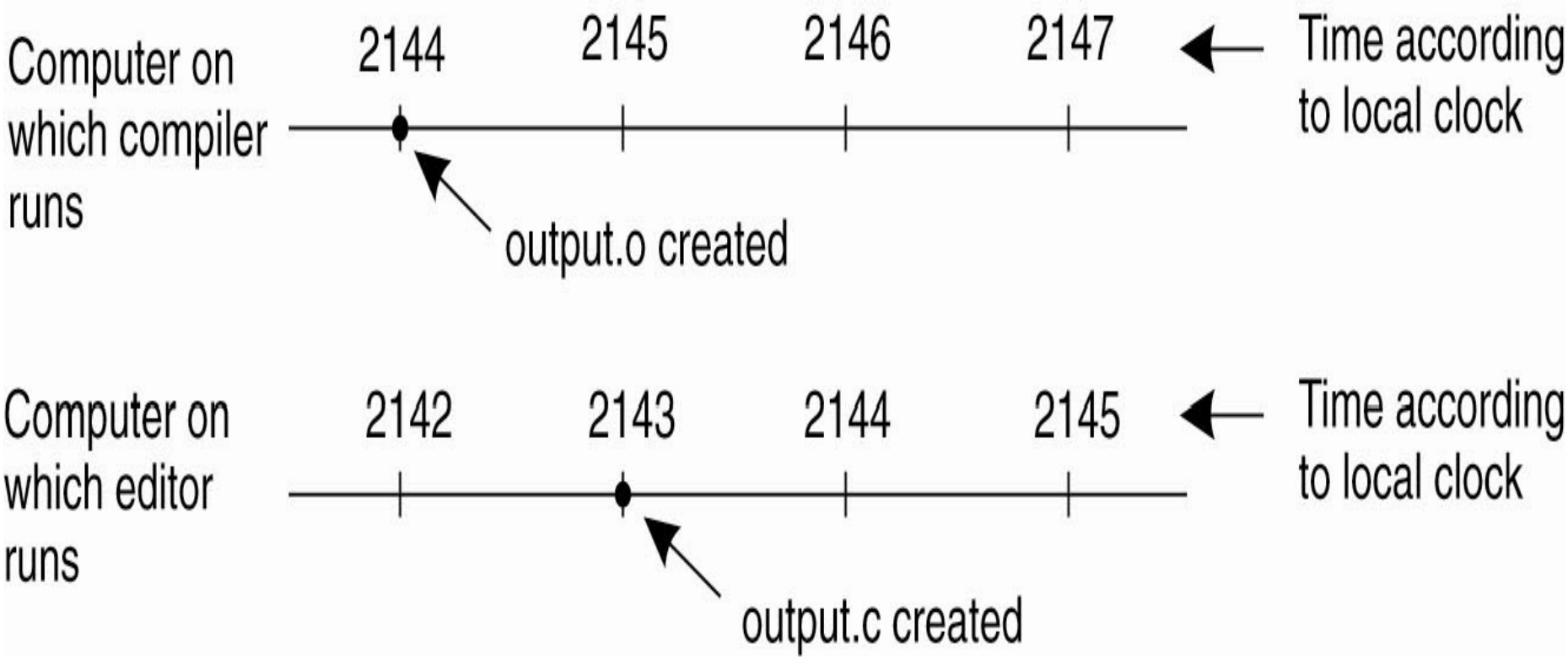
وقتی برنامه نویس تغییر در تمام فایل های مبدأ را به اتمام رساند، برنامه **make** را اجرا می نماید. این برنامه، با بررسی زمان اصلاح تمام فایل های مبدأ **source** و مقصد **object** متوجه می شود کدام فایل مبدا نیاز به ترجمه مجدد دارد.

# همگام سازی ساعت (۲)

اگر فایل مبدأ **input.c** دارای زمان ۲۱۵۱ و فایل مقصد آن، **input.o** دارای زمان ۲۱۵۰ باشد، **make** متوجه می شود که **input.c** پس از ایجاد **input.o** تغییر کرده است و در نتیجه **input.c** باید دوباره ترجمه شود.

از طرف دیگر، اگر **input.c** دارای زمان کمتری از **input.o** باشد نیاز به ترجمه مجدد نیست.

# همگام سازی ساعت (۳)




**وقتی هر ماشین دارای ساعت خاص خودش است، رویدادی که پس از رویداد دیگری اتفاق افتاد، ممکن است زمان زودتری به آن نسبت داده شود.**

# همگام سازی ساعت (۴)

۱- فیزیکی :

واقعاً همه ساعت های کامپیوترها یکسان می شوند.

۲- منطقی :

لزومی ندارد که ساعت همه یکسان باشد ممکن است ساعت هایشان کم و زیاد باشد ولی یک جوری است که کارشان به درستی انجام می شود.

# همگام سازی ساعت (۵)

**Clock Tick** (تیک ساعت):

هر دفعه یک **Interrupt** رخ می دهد که باعث می شود شمارنده ساعت یک واحد افزایش یابد.

یک **Counter** برای ساعت وجود دارد که به آن **Holding Register** نیز می گویند و کارش این است که ساعت را نشان می دهد.

# ساعت های فیزیکی (۱)

تمام کامپیوتر ها داری مداری برای نگهداری زمان هستند (تایمر).

با یک کامپیوتر و یک ساعت، مهم نیست که این ساعت برای مدتی خاموش شود. چون تمام فرآیند ها در ماشین از یک ساعت استفاده می کنند، از نظر داخلی نیز سازگار خواهند بود.

به محض معرفی چندین کامپیوتر که هر کدام ساعت مخصوص به خود را داشتند، وضعیت کاملاً تغییر کرد.

# ساعت های فیزیکی (۲)

وقتی شبکه دارای n کامپیوتر باشد، تمام n کریستال تا حدودی با نرخ های متفاوت اجرا می شوند و به این ترتیب ساعت ها به تدریج از همگامی خارج می شوند و مقادیر متفاوتی را ارائه می کنند.

این تفاوت در مقادیر زمان، **انحراف ساعت (Clock drift)** نام دارد.

به دلایل افزایش کارایی و افزونگی، وجود چندین ساعت فیزیکی مطلوب است که منجر به دو مسئله می شود:

۱- چگونه آن ها را با ساعت های دنیای واقعی همگام کنیم

۲- چگونه ساعت ها را با یکدیگر همگام کنیم

# ساعت فیزیکی (۳)-ساعت خورشیدی

- Some Basic definitions:
- Transit of the Sun:
  - Sun rises from east
  - Rises to its max height in the sky (noon)
  - Sun sets in west
- Solar Day: Interval between 2 consecutive transits of the sun (noon-to-noon)
- Solar Day varies due:
  - Core activities of earth
  - Rise & Tide of oceans
  - Gravity
  - Orbit around the sun, not a perfect circle

# ساعت فیزیکی (۳) – ساعت اتمی

با اختراع ساعت اتمی در سال ۱۹۴۸، مستقل از حرکت و لرزش زمین، اندازه گیری دقیق تر ساعت امکان پذیر شد.

TAI (International Atomic Time) فقط تعداد میانگین تیک های ساعت ۱۳۳ سزیم از نیمه شب ۱۹۵۸، ۱Jan، (شروع زمان) تقسیم بر ۹۱۹۲۶۳۱۷۷۰ است.

زمانی بر اساس ثانیه های TAI به نام زمان هماهنگ شده ی جهانی یا UTC خوانده می شود. UTC مبنایی برای تمام ساعت های مدرن است.

UTC جایگزین استاندارد های قدیمی، یعنی زمان میانگین گرینویچ شد که زمان اختر شناسی است.

# ساعت های فیزیکی (۴)

کار UTC ایجاد و ارسال پالس ساعت بود و کافی بود هر کسی یک wwv receiver داشته باشد.

GMT ساعت استاندارد جهانی از نظر فیزیکی است.

UTC خوب بود ولی امروزه همه ترجیح می دهند از GMT استفاده کنند.

# سیستم تعیین موقعیت جهانی

**مسئله تعیین موقعیت، از طریق سیستم توزیع شده خاصی به نام GPS حل می شود (GPS سیستم ماهواره ای است).**

**GPS از ۲۹ ماهواره استفاده می کند که ماهواره ها بطور پیوسته موقعیت خود را پخش می کنند. و هرکدام در ارتفاع تقریبی ۲۰۰۰۰ کیلومتری می چرخند. هر ماهواره تا چهار ساعت اتمی دارد، که دائماً از ایستگاههای خاصی در زمین تنظیم می شوند. نیاز به حداقل سه ماهواره برای تعیین طول, عرض و ارتفاع یک موقعیت**

**دو حقیقت در دنیای واقعی وجود دارند که باید در نظر گرفته شوند:**

**۱- قبل از این که داده های موجود در موقعیت ماهواره به گیرنده برسند، مدتی زمان صرف می شود. ۲- ساعت گیرنده با ساعت ماهواره همگام نیست.**

# الگوریتم های همگام سازی ساعت (۱)

اگر یک ماشین دارای گیرنده WWV باشد، هدف این است که تمام ماشین های دیگر با آن همگام شوند.

اگر هیچ ماشینی دارای گیرنده WWV نباشد، هر ماشین زمان خاص خودش را نگهداری می کند.

الگوریتم های متعددی برای انجام این همگام سازی پیشنهاد شدند.

# الگوریتم های همگام سازی ساعت (۲)

- اگر ثابتی مثل مثل p وجود داشته باشد که:

$$۱-p <= \frac{DC}{Dt} <= ۱+P$$

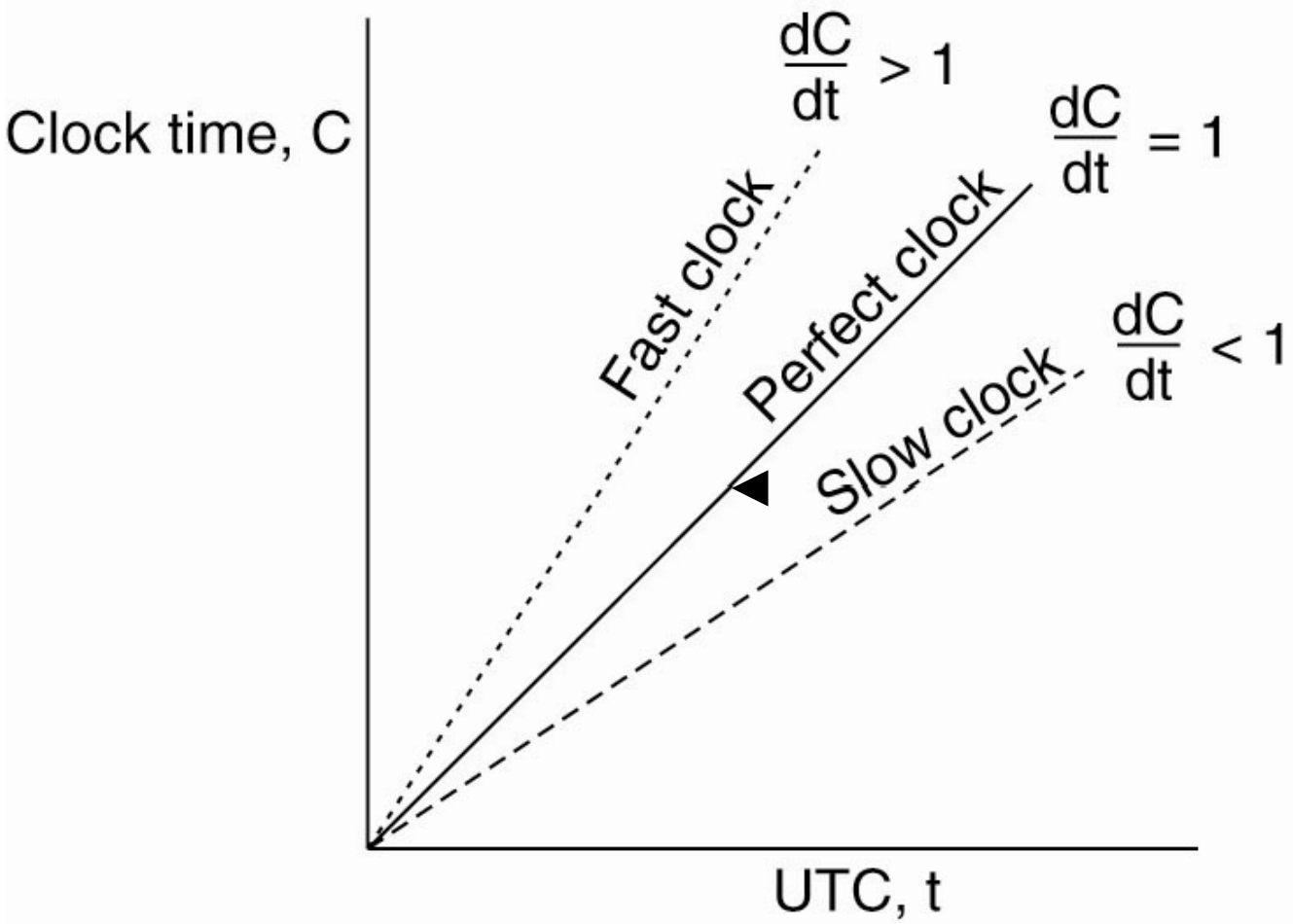
می توان گفت که ساعت در مشخصات خودش کار می کند.

P توسط کارخانه مشخص می شود و حداکثر Drift rate نام دارد.

DC تفاضل ساعت ماشین

Dt تفاضل ساعت UTC

# الگوریتم های همگام سازی ساعت (۳)




**رابطه بین زمان ساعت و UTC ، وقتی که ساعت ها با نرخ های متفاوتی تیک می زنند.**

# الگوریتم های همگام سازی ساعت (۴)

در اینترنت از GMT استفاده می شود.

استفاده از Time Server یکی از روش هایی است که برای همگام سازی ساعت استفاده می شود.

اشکال این روش این است که Event Base است یعنی وقتی که ما به اینترنت وصل می شویم این کار را انجام می دهد.

# الگوریتم های همگام سازی ساعت (۵)

۱- الگوریتم پروتکل زمان شبکه (NTP)

۲- الگوریتم برکلی (Berkeley)

۳- الگوریتم همگام سازی ساعت در شبکه های بی سیم(RBS)

# الگوریتم زمان شبکه (NTP)

یک روش متداول در بسیاری از پروتکل ها که توسط کریستین(۱۹۸۹) پیشنهاد شد. این است که به سرویس گیرنده ها اجازه داده شود که با سرویس دهنده ساعت تماس بگیرند.

سرویس دهنده می تواند زمان فعلی را به طور دقیق فراهم کند.

در سیستم توزیع شده یک کامپیوتر (سرویس دهنده) هست که ساعتش **Update** است (دارای گیرنده **WWV** است).

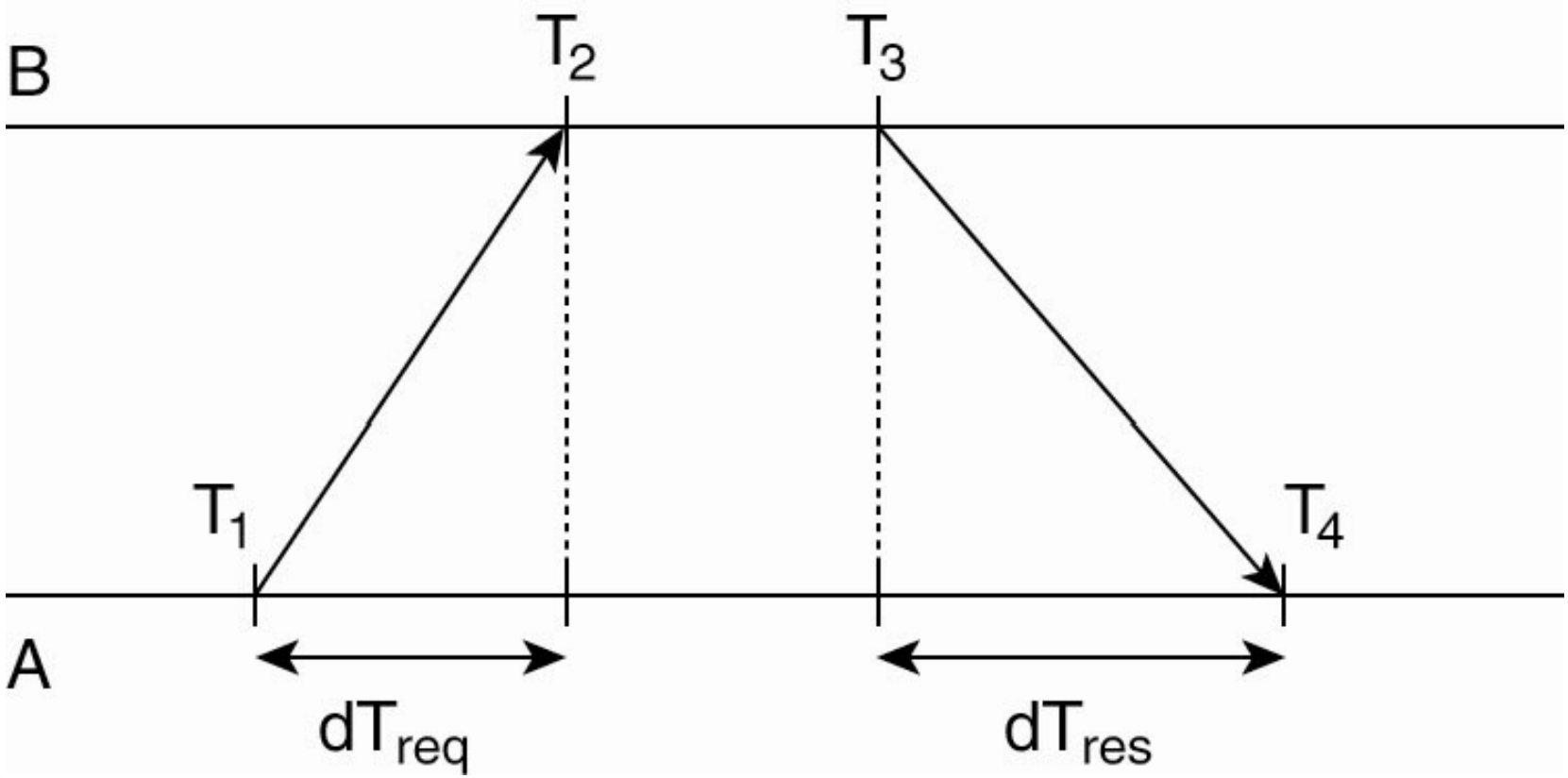
# الگوریتم زمان شبکه (NTP)

**کاری که انجام می شود این است که هر کامپیوتر در فواصل زمانی خاصی که دوست ندارد بیش از آن Drift rate اش بالاتر برود از کامپیوتر سرویس دهنده زمان درخواست می کند و پاسخ را دریافت می کند.**

**مشکل ۱: زمان به گذشته باز نمی گردد.**

**مشکل ۲: هنگام تماس با سرویس دهنده، تأخیر های پیام، زمان گزارش شده را قدیمی می کند.**

# حل مشکل ۲ در الگوریتم زمان شبکه (NTP)




**دریافت زمان فعلی توسط A از سرویس دهنده زمان B**

# حل مشکل ۲ در الگوریتم زمان شبکه (NTP)

در این مورد A درخواستی را به B می فرستد که مقدار مهر زمان ارسال آن T1 است.

B زمان رسیدن درخواست یعنی T2 را ثبت می کند و پاسخی را می فرستد که مهر زمان ارسال آن T3 است، سرانجام A زمان رسیدن پاسخ یعنی T4 را ذخیره می کند. A می تواند آفست خود را نسبت به B به صورت زیر محاسبه کند:

$$\theta = T_3 - \frac{(T_2 - T_1) + (T_4 - T_3)}{2} = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

اگر کلاک A سریعتر باشد آنگاه $\theta < 0$ یعنی ساعت A بایستی به عقب برگردد!

مقدار تاخیر بر اساس پروتکل NTP بین A و B برابر است با:

$$\delta = \frac{(T2-T1) + (T4-T3)}{2} = \frac{(T4-T1)}{2}$$

# الگوریتم زمان شبکه (NTP)

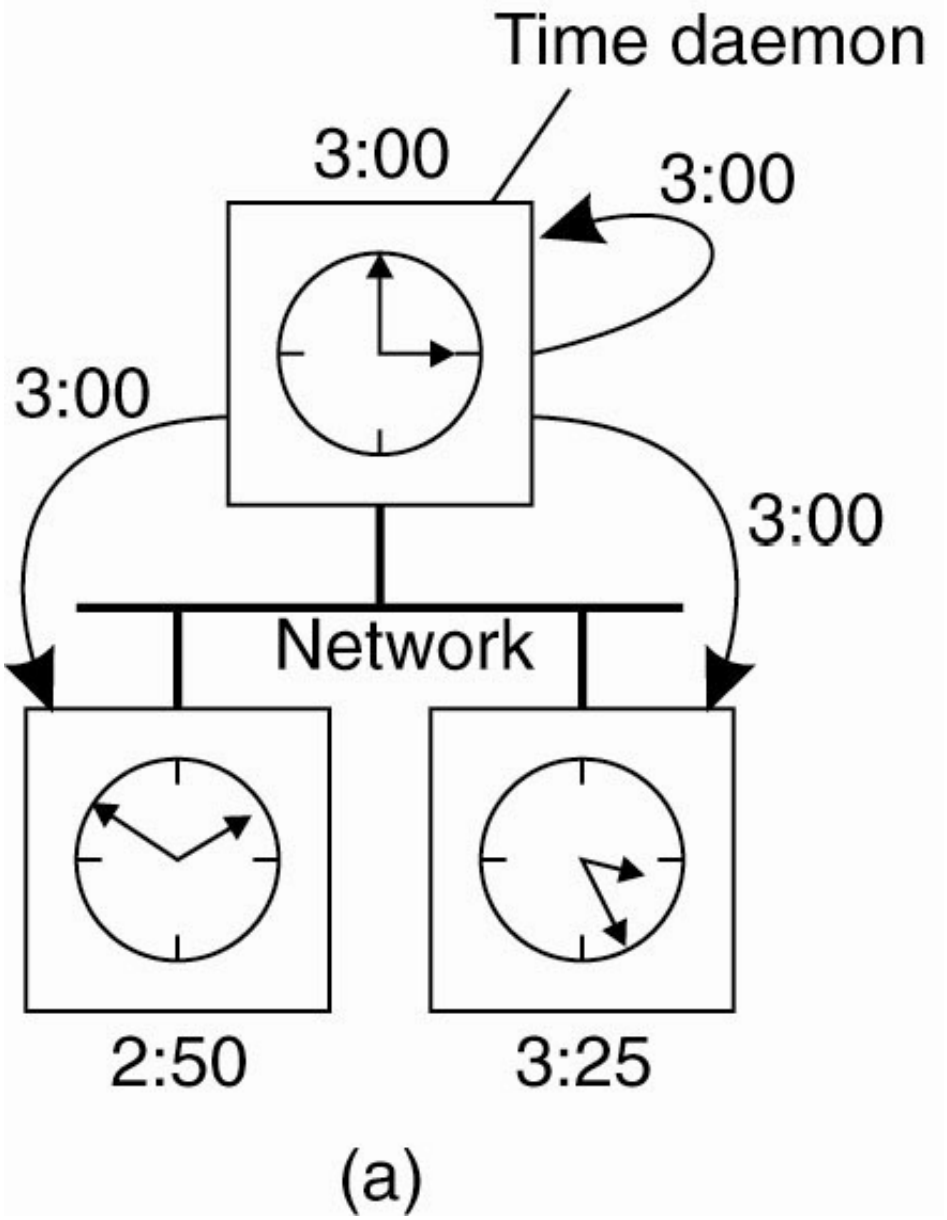
**در NTP ، سرویس دهنده زمان(Time Server) غیر فعال است یعنی شروع کننده نیستند و سایر ماشین ها به طور دوره ای زمان را از آن می پرسند و آن نیز پاسخ می دهد (Client فعال و همیشه شروع کننده است).**

# الگوریتم برکلی(Berkeley)

در اینجا سرویس دهنده زمان فعال است به طوری که در فواصل منظم از هر ماشین، زمان را می پرسد. بر اساس پاسخ ها، میانگین (میانه) را محاسبه می کند و به تمام ماشین ها می گوید ساعت های خود را به زمان جدید جلو ببرند (با سریع کردن تیک ساعت) یا ساعت خود را به عقب بکشند! (با کند کردن تیک ساعت).

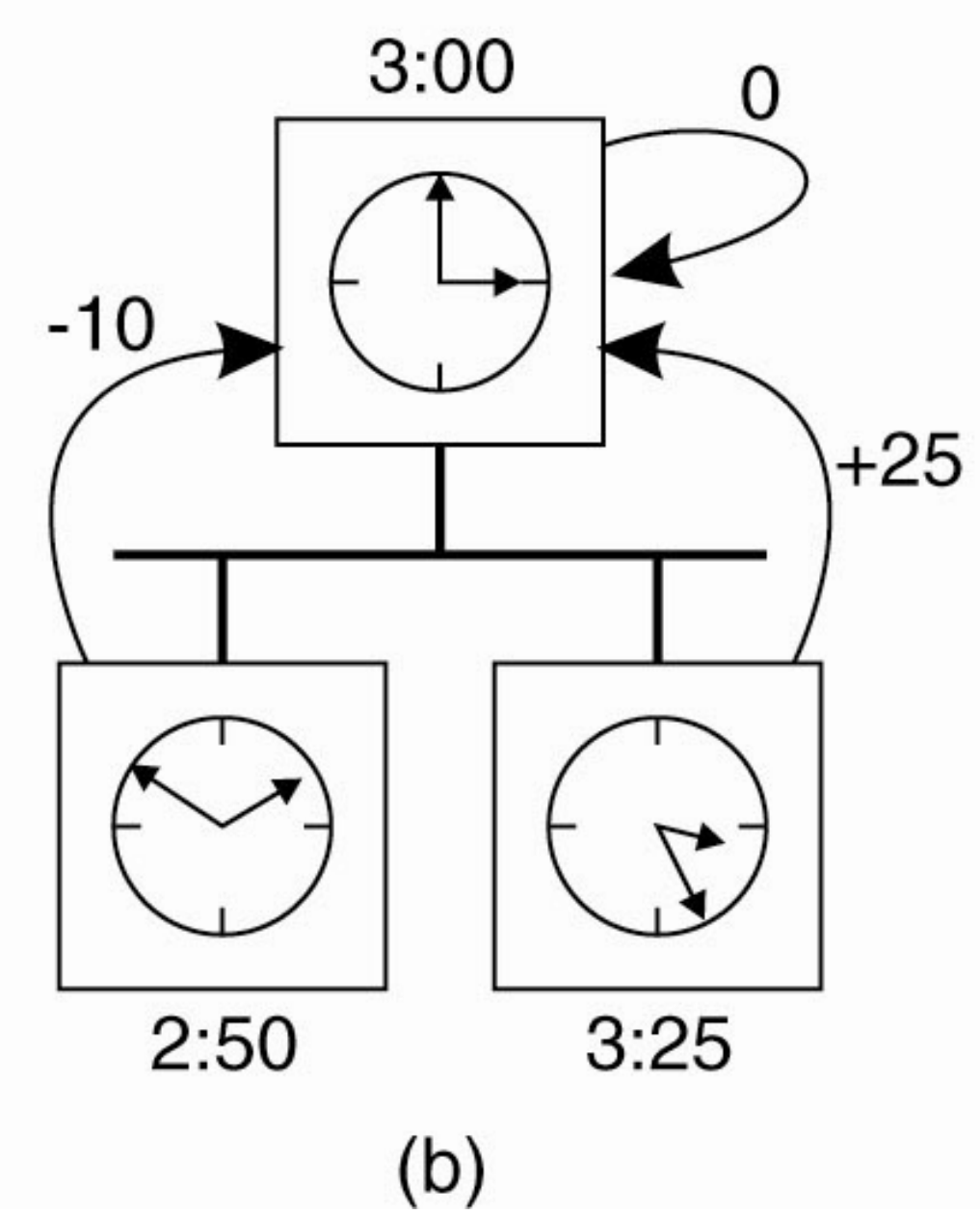
این الگوریتم برای سیستمی مفید است که در آن، هیچ ماشینی، گیرنده WWV ندارد.

# الگوریتم برکلی(Berkeley) – ادامه



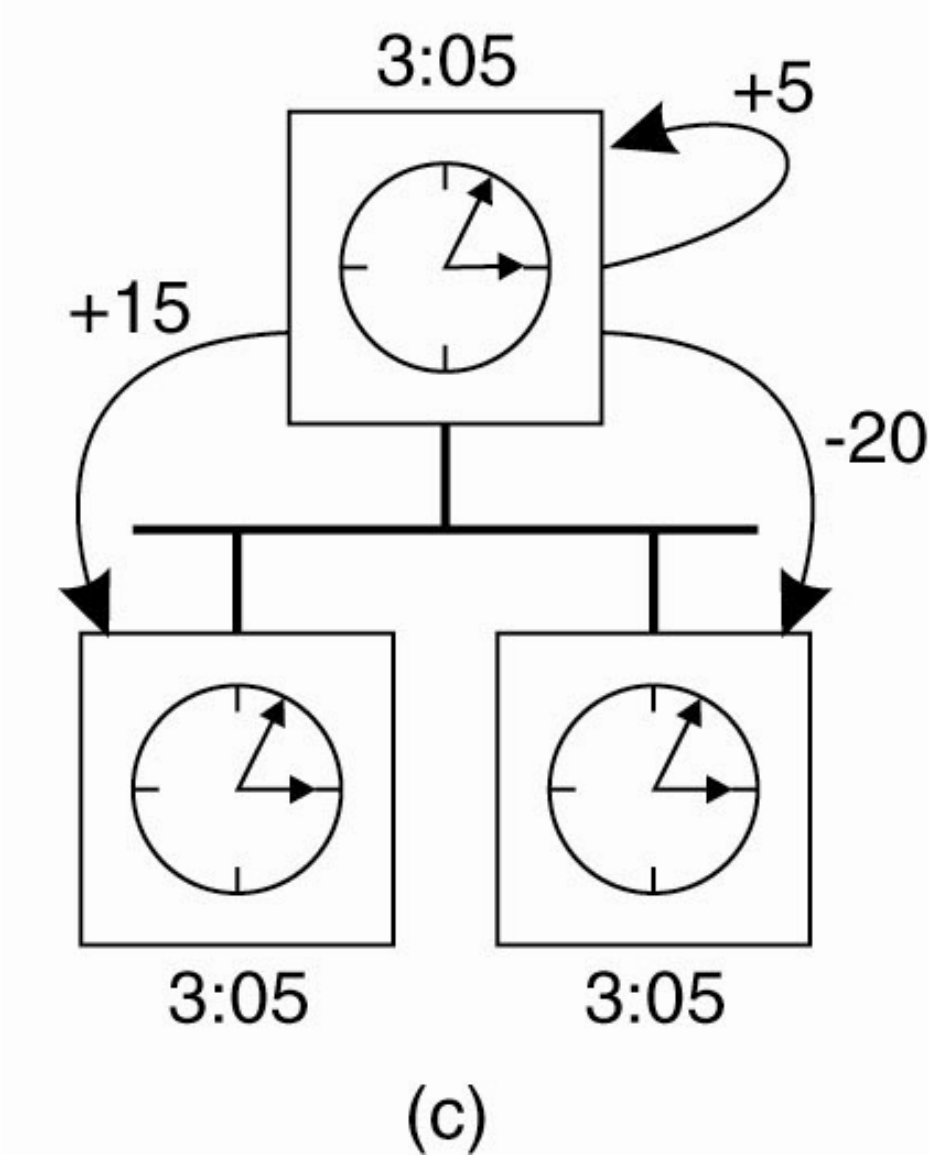

a) دمون زمان، مقادیر ساعت تمام ماشین ها را سوال می کند.

# الگوریتم برکلی(Berkeley) – ادامه




b) ماشین ها پاسخ می دهند.

# الگوریتم برکلی(Berkeley) – ادامه




C) دمون زمان به هر ماشین می گوید که ساعتش را چگونه تنظیم کند.

# الگوریتم برکلی(Berkeley) – ادامه

لازم نیست که این زمان با زمان واقعی که هر ساعت توسط رادیو اعلان می شود یکسان باشد.

الگوریتم برکلی مستقل از کامپایلر و زبان است و در سیستم های ناهمگن نسبت به NTP مناسب تر است.

# ساعت های منطقی

لامپورت نشان داد که گر چه همگام سازی ساعت امکان پذیر است، لازم نیست مطلق باشد. اگر دو فرآیند تعامل نداشته باشند، لازم نیست ساعت های آن ها همگام شوند، زیرا مشکلی به وجود نمی آید.

او اشاره کرد که مهم نیست تمام فرآیند ها بر زمان خاصی توافق داشته باشند، بلکه بر روی ترتیب وقوع رویدادها توافق دارند.

# ساعت منطقی لامپورت (۱)

**برای همگام سازی ساعت های منطقی، لامپورت رابطه ای به نام** تقدم رویداد **happens-before (→) را با شرایط زیر تعریف کرد:**

**۱- در یک ماشین یا فرایند اگر a قبل از b رخ دهد آنگاه $a \rightarrow b$ درست است.**

**۲- در دو ماشین یا فرایند پیام نمی تواند قبل از ارسال، دریافت شود، یا حتی همزمان با ارسال، دریافت شود، زیرا مدتی زمان نیاز دارد تا به گیرنده برسد.**

**عبارت $a \rightarrow b$ به این صورت خوانده می شود: a قبل از b رخ می دهد.**

# ساعت منطقی لامپورت (۲)

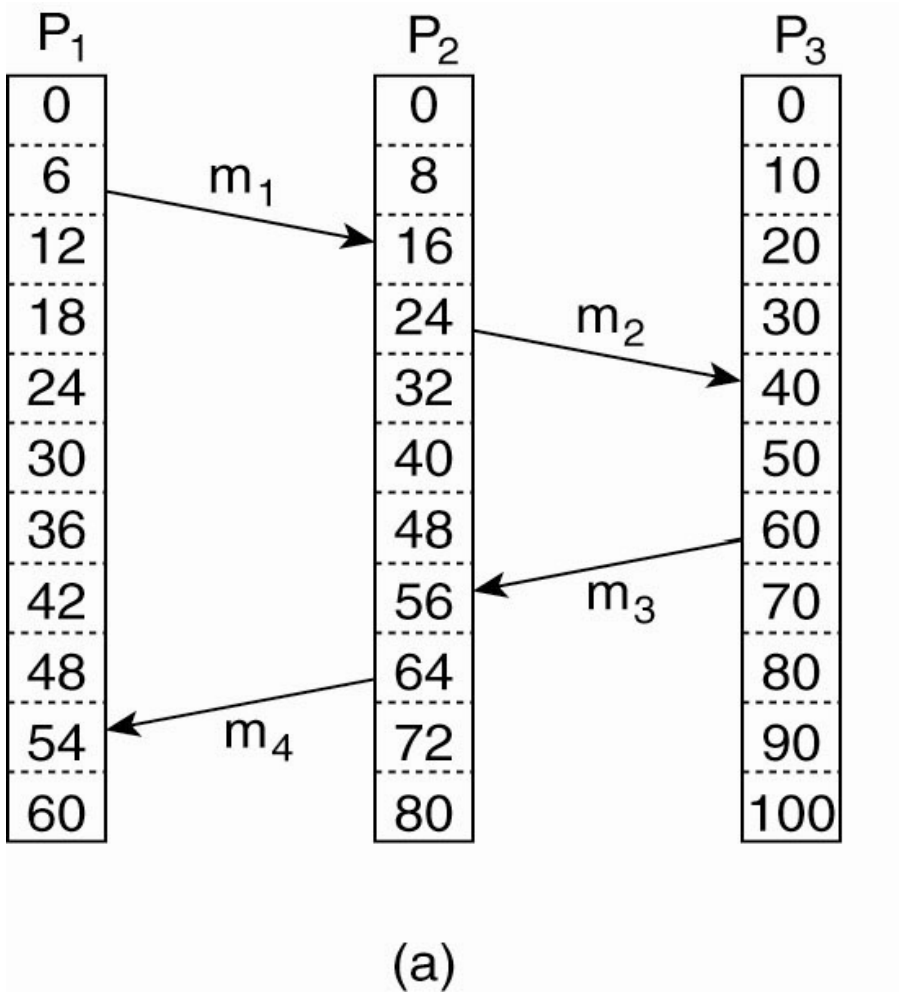
تقدم رویداد، رابطه تراگذری است:

لذا اگر $a \rightarrow b$ و $b \rightarrow c$ آنگاه $a \rightarrow c$.

ساعت در این روش هیچ گاه به عقب بر نمی گردد و تنها به جلو افزایش می یابد، یعنی ساعت هیچ گاه منفی نمی شود.

سیستمی که ساعتش از همه بیشتر است اگر برای دیگران پیغام بفرستد ساعت همه را بالا می برد.

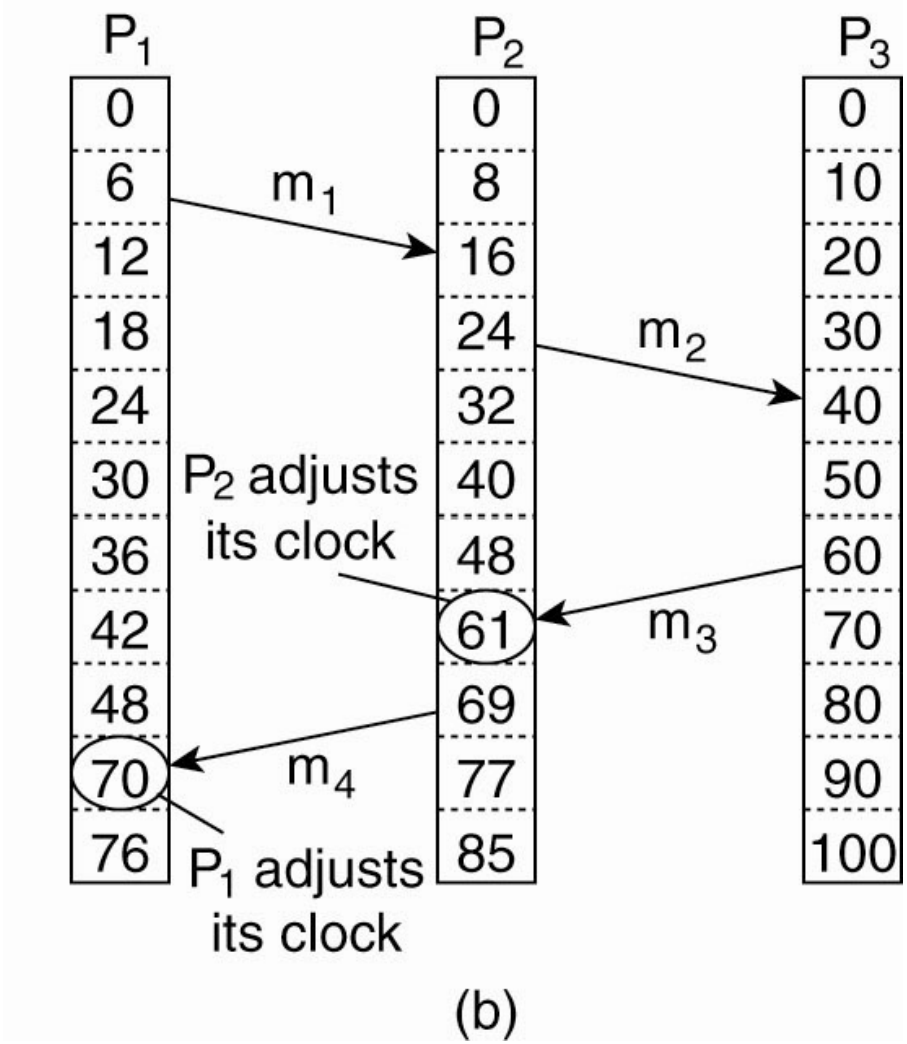
# ساعت منطقی لامپورت - مثال (۱)




(a)

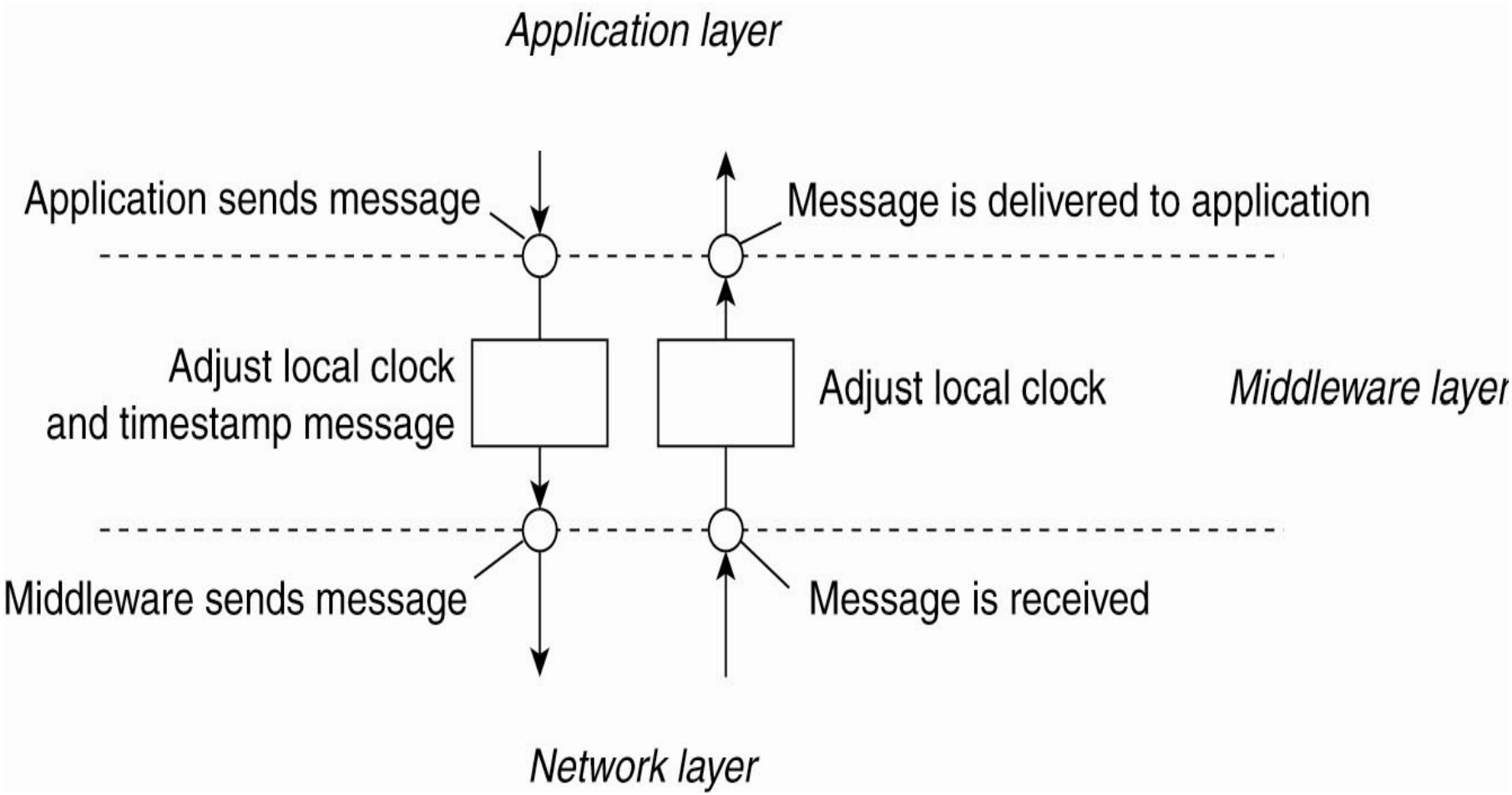
**سه فرآیند که هر کدام ساعت خاص خودش را دارد. ساعت ها با تیکهای مختلفی اجرا می شوند.**

# ساعت منطقی لامپورت - مثال (۲)




**الگوریتم لامپورت ساعت ها را اصلاح می کند.**

# ساعت منطقی لامپورت - ادامه




استقرار ساعت های منطقی در سیستم های توزیع شده.

# ساعت منطقی لامپورت - ادامه

هر فرآیند P(i) یک شمارنده محلی c(i) را نگهداری می کند. این شمارنده ها به صورت زیر به هنگام می شوند:

۱- قبل از اجرای رویداد P(i) رابطه c(i)←c(i) +۱ را اجرا می کند.

۲- وقتی فرآیند P(i) پیام m را به P(j) می فرستد، مهر زمانی m، یعنی ts(m) را پس از اجرای مرحله اول، برابر با c(i) قرار می دهد

۳- به محض دریافت پیام m، فرآیند P(j) شمارنده محلی خود را به صورت c(i) ← max{c(j),ts(m)} تنظیم می کند که پس از آن مرحله اول را اجرا می کند و پیام را به کاربرد تحویل می دهد.

# ساعت های برداری

ساعت های لامپورت علّیت را در نظر نمی گیرند.

با استفاده از ساعت های برداری می توان تضمین کرد که پیام وقتی تحویل داده می شود که تمام پیام هایی که از نظر علّیتی قبل از آن وجود دارند، دریافت شدند.

علیت را می توان بوسیله ساعت های برداری در نظر گرفت.

ساعت برداری VC(a) که به رویداد a تخصیص می یابد دارای این خاصیت است که اگر برای رویدادی مثل b ، داشته باشیم

VC(a) < VC(b) ، آن گاه می گوییم رویداد a از نظر علیتی قبل از b قرار دارد.

# همگام سازی فرآیند ها(Process)

برای همگام سازی فرآیند ها بحث دوبدوناسازگاری یا انحصار متقابل Mutual Exclusion مطرح می شود.

بخش بحرانی نه همزمان نه همروند در اختیار ۲ تا فرآیند نباشد.

باید اجازه دهیم فقط یک فرآیند از بخش بحرانی استفاده کند در صورتی فرآیند بعد می تواند از بخش بحرانی استفاده کند که فرآیند اول کارش تمام شده باشد.

# الگوریتم های انحصار متقابل

**الگوریتم های انحصار متقابل توزیع شده:**

**۱- روش مبتنی بر نشانه (Token based)**

**۲- روش مبتنی بر اجازه (Permission Based)**

# روش مبتنی بر نشانه

از طریق ارسال پیام خاصی بین فرآیند ها انجام می شود که نشانه نام دارد.

فقط یک نشانه وجود دارد و هر کسی که آن نشانه را دارد می تواند به منبع مشترک دستیابی داشته باشد. پس از اتمام دستیابی، نشانه به فرآیند بعدی فرستاده می شود.

این روش جلوگیری از بن بست را تضمین می کند.

عیب عمده این روش این است که، وقتی نشانه مفقود می شود، باید رویه توزیع شده ی دقیقی اجرا شود تا تضمین گردد که نشانه جدیدی ایجاد شده است و این نشانه یکتا است.

# روش مبتنی بر اجازه

در این روش فرآیندی که منتظر دستیابی به منبع است، ابتدا نیاز به اجازه یک فرآیند هماهنگ کننده یا سایر فرآیند ها دارد.

انواع روشهایی که برای اعطای چنین اجازه ای وجود دارد:

۱- الگوریتم متمرکز

۲- الگوریتم نامتمرکز

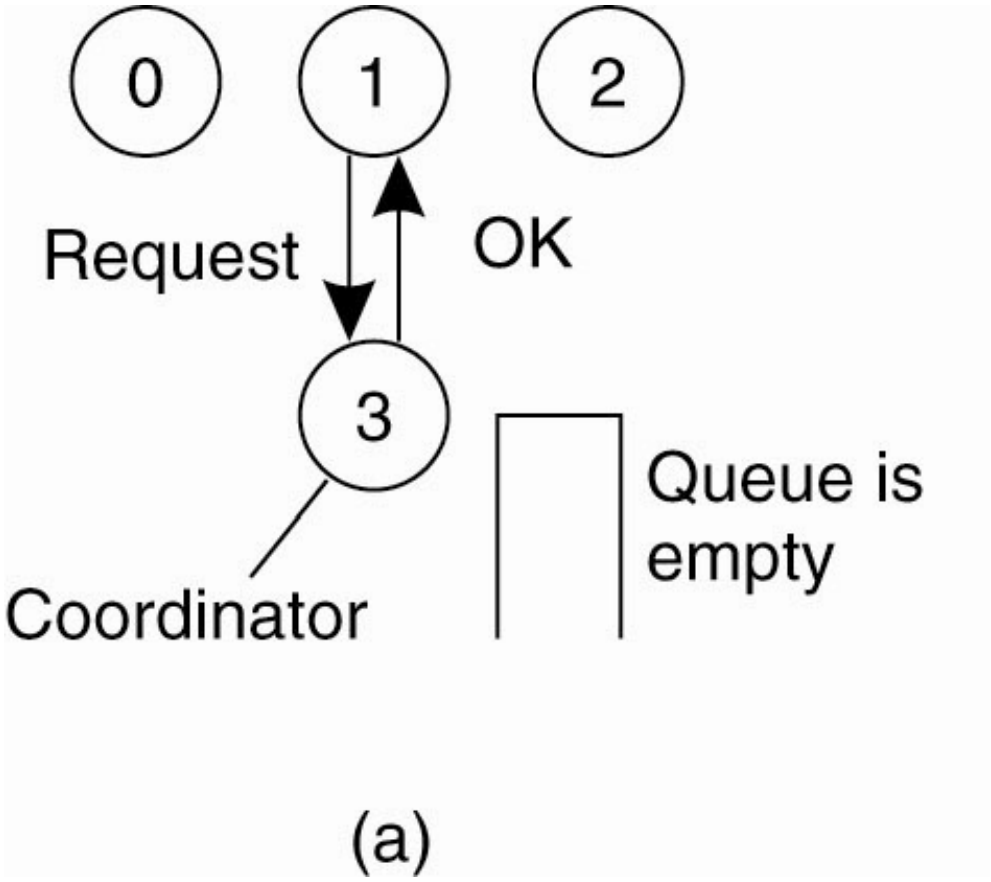
۳- الگوریتم توزیع شده

# الگوریتم متمرکز (۱)

راحت ترین راه برای رسیدن به انحصار متقابل در سیستم توزیع شده، شبیه سازی چگونگی انجام آن در سیستم تک پردازنده ای است.

یک فرآیند به عنوان هماهنگ کننده انتخاب می شود. هر وقت فرآیندی می خواهد به منبع مشترکی دست یابد، پیام درخواست را به هماهنگ کننده می فرستد و می گوید به کدام منبع می خواهد دستیابی داشته باشد و کسب اجازه می کند.

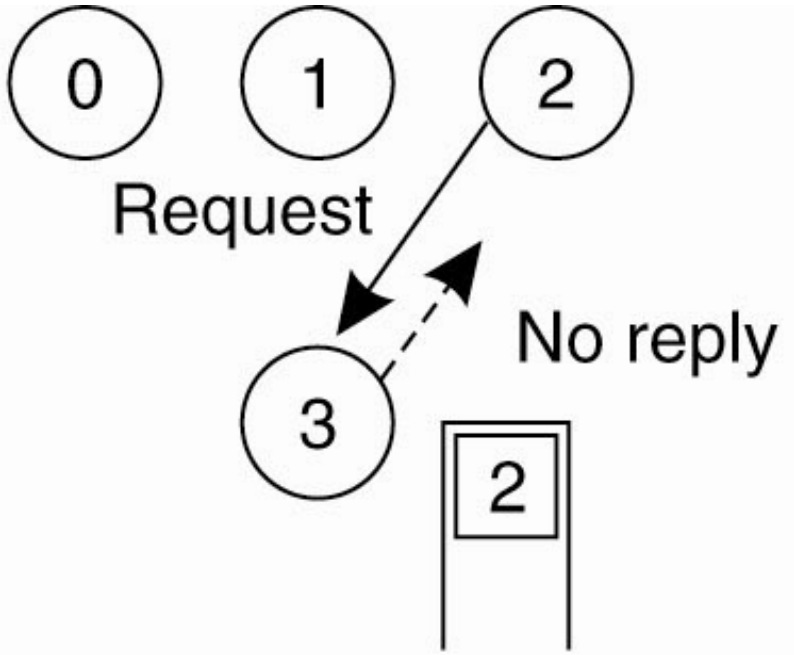
# الگوریتم متمرکز (۲)




(a)

فرآیند ۱ از هماهنگ کننده اجازه می خواهد تا به منبع مشترک دست یابد. سپس هماهنگ کننده اجازه می دهد.

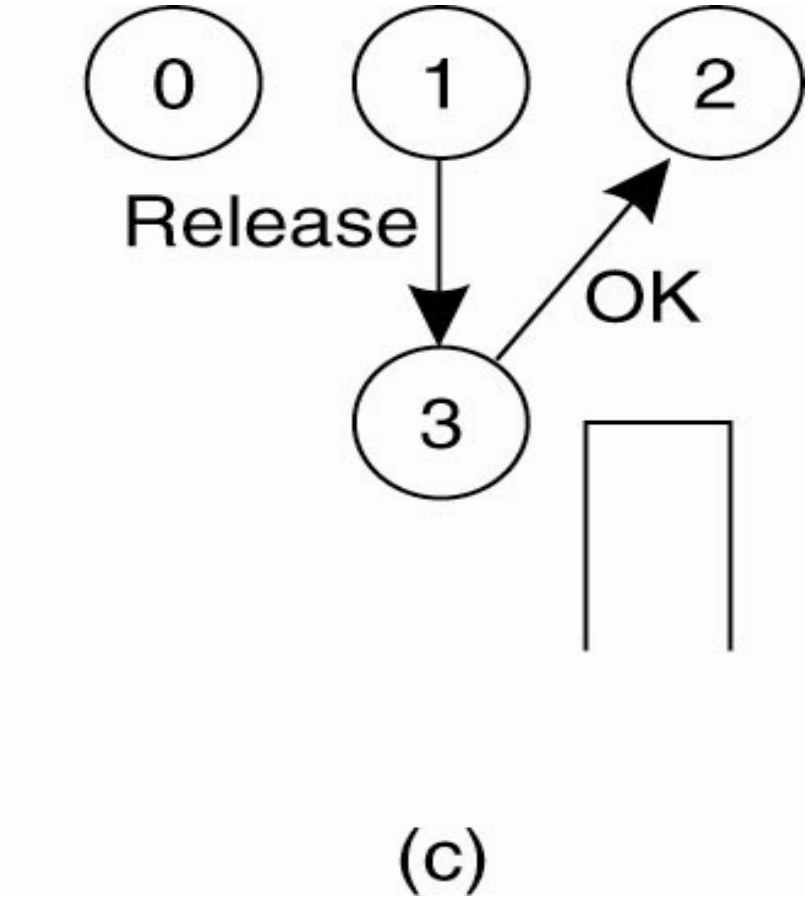
# الگوریتم متمرکز (۳)




(b)

فرآیند ۲ اجازه می خواهد تا به همان منبع دست یابد. هماهنگ کننده پاسخ نمی دهد.

# الگوریتم متمرکز (۴)




(c)

وقتی فرآیند ۱ منبع را آزاد می کند، به هماهنگ کننده اعلان می کند، که آن نیز به فرآیند ۲ پاسخ دهد.

# ویژگی های الگوریتم متمرکز

**پیاده سازی** و **مدیریت** آن خیلی ساده است. **انحصار متقابل** را تضمین می کند.

**گرسنگی** وجود ندارد.

هماهنگ کننده، **یک نقطه شکست** دارد واگر خراب شود، کل سیستم از کار می افتد.

تنها یک هماهنگ کننده می تواند **گلوگاه** برای کارایی محسوب شود.

# الگوریتم نامتمرکز

یک روش رأی گیری مبتنی بر سیستم DHT است و روش آن این است که هماهنگ کننده مرکزی را بسط می دهد.

- فرض می شود هر منبع n بار تکثیر شده است. هر کپی دارای هماهنگ کننده خاص خودش برای کنترل دستیابی توسط فرآیندهای همزمان است.
- هر وقت فرآیندی می خواهد به منبعی دستیابی داشته باشد،لازم است از **m>n/2** هماهنگ کننده رأی جمع کند.
- اگر فرایندی کمتر از m رای جمع نماید. دوباره بعد از زمان تصادفی همان کار را تکرار می کند.
- در صورت خرابی هماهنگ کننده و ترمیم آن کلیه رایهای داده شده را فراموش می کند که احتمال وقوع مشکل حتی در صورت رای دادن به فرایند جدید در صورتی که قبلا به دیگری هم رای داده باشد بسیار پایین است.

# ویژگی های الگوریتم نامتمرکز

نسبت به الگوریتم متمرکز کمتر دچار آسیب پذیری می شود.

یک نقطه شکست وجود ندارد.

گلوگاه وجود ندارد (صف وجود ندارد).

# الگوریتم توزیع شده (۱)

وقتی فرآیندی می خواهد به منبع مشترکی دست یابد، پیامی می سازد که شامل نام منبع، شماره فرآیند آن و زمان (منطقی) فعلی است. سپس پیام را به تمام فرآیند ها گروه خودش ارسال می کند.

وقتی فرآیندی پیام درخواست را از فرآیند دیگری دریافت می کند، یکی از سه حالت مختلف آینده را انجام می دهد:

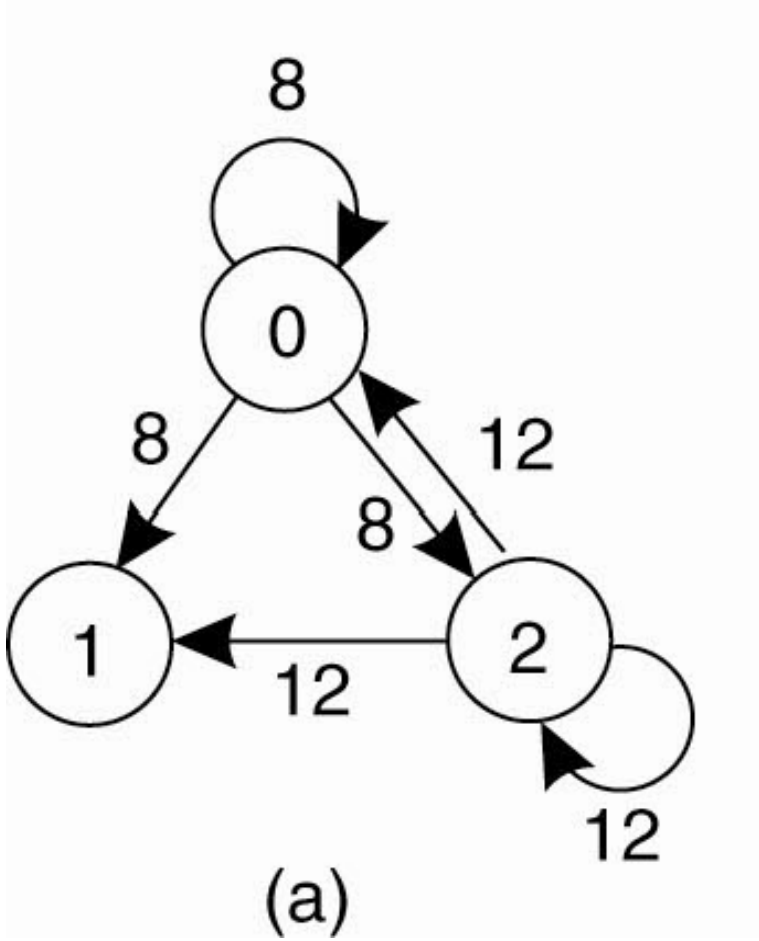
# الگوریتم توزیع شده (۲)

۱- اگر گیرنده در حال دستیابی به منبع نباشد و نخواهد به آن دست یابد، پیام **OK** را به فرستنده ارسال می کند.

۲- اگر گیرنده به منبع دستیابی داشته باشد، در عوض، درخواست را در صف قرار می دهد.

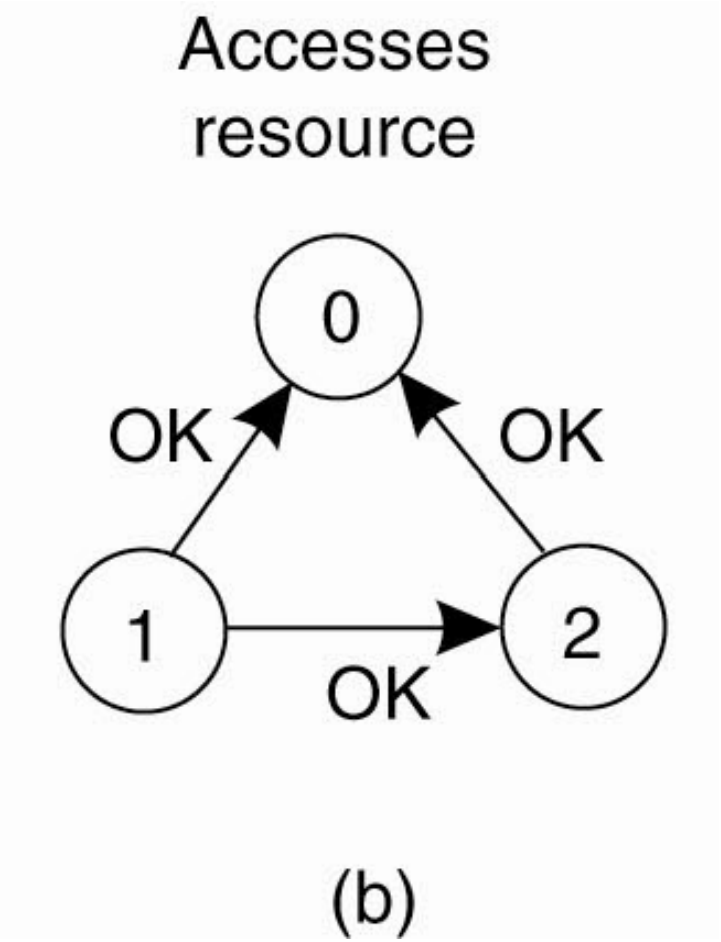
۳- اگر گیرنده بخواهد به منبع دستیابی داشته باشد ولی هنوز موفق نشده است، مهر زمان پیام ورودی را با مهر زمانی موجود در پیامی که به هر کسی فرستاده است، مقایسه می کند. کمترین مهر زمانی برنده است.

# الگوریتم توزیع شده (۳)



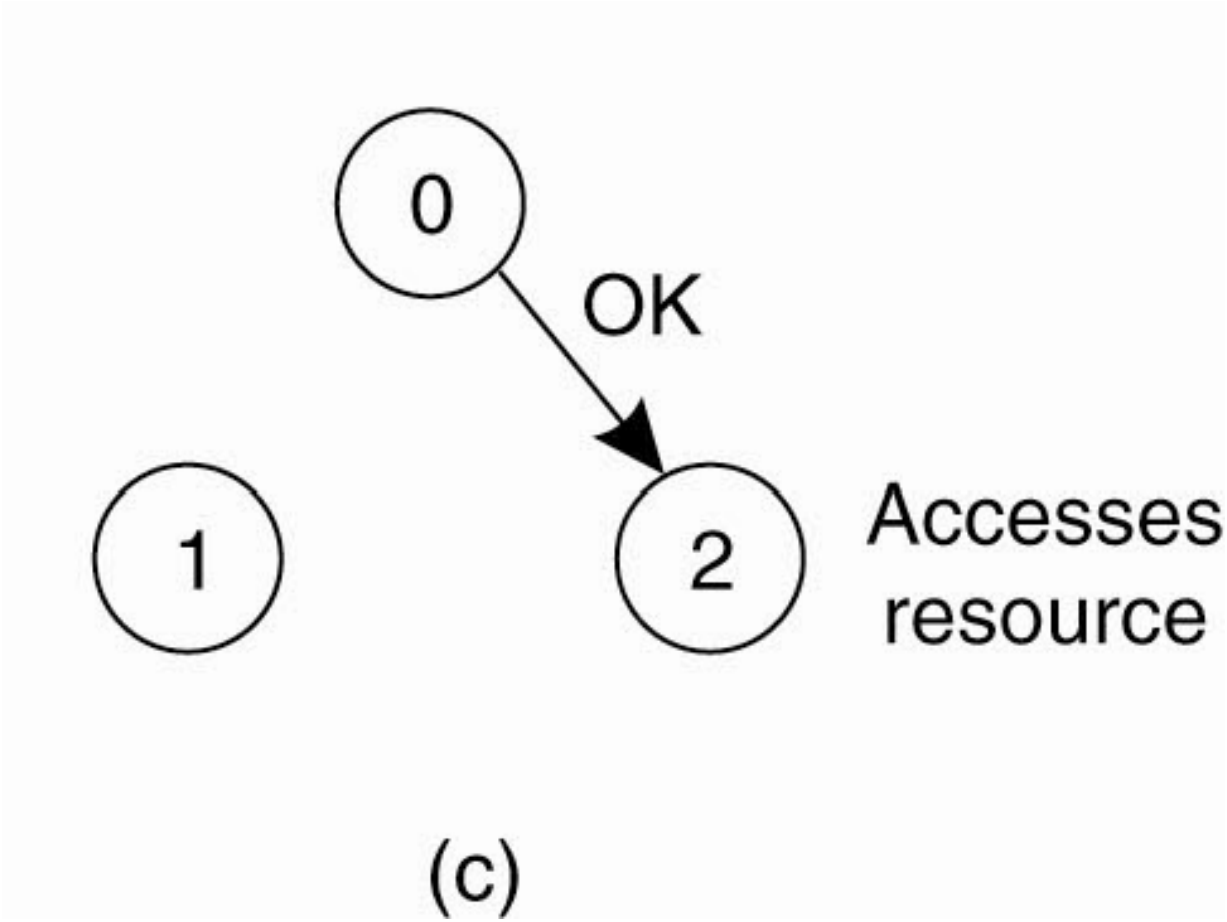

**دو فرآیند می خواهند همزمان به منبع مشترک دستیابی داشته باشند.**

# الگوریتم توزیع شده (۴)




فرآیند صفر مهر زمان کمتری دارد، لذا برنده است.

# الگوریتم توزیع شده (۵)




وقتی فرآیند صفر کارش را انجام داد، پیام OK را می فرستد و فرآیند ۲ می تواند به کارش ادامه دهد.

# ویژگی های الگوریتم توزیع شده

سوال: آیا همواره الگوریتم توزیع شده از الگوریتم متمرکز بهتر است؟

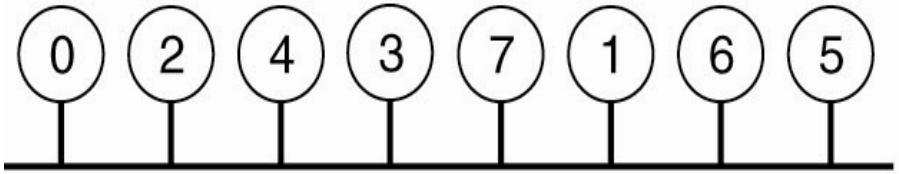
همانند الگوریتم متمرکز، انحصار متقابل بدون بن بست و گرسنگی تضمین می شود.

n نقطه شکست وجود دارد! و اگر یکی از کار بیفتد همه از کار می افتند.

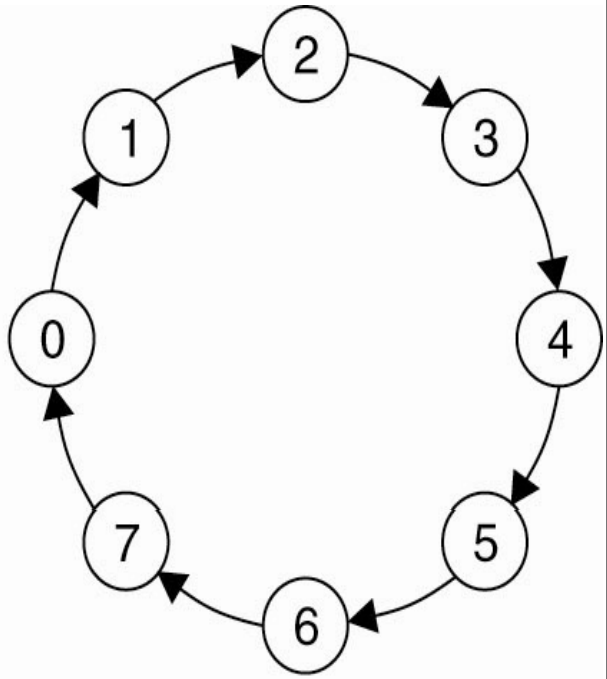
حجم پیام ها خیلی بالا است.

نسبت به الگوریتم متمرکز کندتر، پیچیده تر، گران تر و با توانمندی کمتر است!

# الگوریتم حلقه نشانه (۱)




(a)




(b)

(a گروه نامرتبی از فرآیند ها در شبکه.

(b حلقه منطقی که در نرم افزار ساخته می شود.

# الگوریتم حلقه نشانه (۲)

در شکل a یک شبکه خطی داریم (مثل اترنت) که فاقد ترتیب فرآیند ها است.

در شکل b به هر فرآیند مکانی در حلقه داده می شود. موقعیت های حلقه ممکن است به ترتیب عددی آدرس های شبکه یا ابزارهای دیگری تخصیص یابد. مهم نیست که ترتیب چه باشد. مهم این است که هر فرآیند می داند بعد از آن چه فرآیندی قرار دارد.

# الگوریتم حلقه نشانه (۳)

وقتی حلقه آماده شد، به فرآیند 0 ، یک نشانه داده می شود. این نشانه در حلقه دور می زند. از فرآیند k به فرآیند k+1 می رود، و بررسی می کند که آیا این فرآیند نیاز به دستیابی به منبع مشترک دارد یا خیر. اگر نیاز داشته باشد، فرآیند اجرا می شود، پس از اتمام کارهایش، منابع را آزاد می کند. پس از پایان کار، نشانه را در ادامه حلقه آزاد می کند.

# ویژگی های الگوریتم حلقه نشانه

در هر زمان فقط یک فرآیند نشانه را در اختیار دارد.

گرسنگی رخ نمی دهد.

اگر نشانه مفقود شود، باید دوباره تولید کرد. (تشخیص مفقود شدن نشانه دشوار است)

اگر فرآیندی خراب شود، الگوریتم دچار درد سر می شود، اما ترمیم آن نسبت به موارد دیگر آسان تر است.

# مقایسه چهار الگوریتم انحصار متقابل

| Algorithm | Messages per entry/exit | Delay before entry (in message times) | Problems |
|---|---|---|---|
| Centralized | 3 | 2 | Coordinator crash |
| Decentralized | 3mk, k = 1,2,... | 2 m | Starvation, low efficiency |
| Distributed | 2 (n – 1) | 2 (n – 1) | Crash of any process |
| Token ring | 1 to ∞ | 0 to n – 1 | Lost token, process crash |

# الگوریتم انتخاب (Election)

در اغلب الگوریتم های توزیع شده لازم است یک فرآیند به عنوان هماهنگ کننده یا آغاز کننده عمل کند.

اگر تمام فرآیند ها دقیقاً یکسان باشند، هیچ راهی برای انتخاب آن ها وجود ندارد.

بطور کلی، الگوریتم انتخاب سعی می کند فرآیندی با شماره بالاتر را به عنوان هماهنگ کننده انتخاب نماید.

# الگوریتم های انتخاب سنتی

۱- الگوریتم توانمند (Bully Algorithm)

۲- الگوریتم حلقه (Ring)

# الگوریتم توانمند (Bully)

همیشه توانمند باقی می ماند.

وقتی فرآیندی می بیند که هماهنگ کننده به درخواست ها پاسخ نمی دهد، انتخاب را شروع می کند.

در هر لحظه، فرآیند می تواند پیام ELECTION را از یکی از همکاران با شماره پایین بگیرد. وقتی چنین پیامی می رسد، گیرنده، پیام OK را به فرستنده ارسال می کند تا نشان دهد که زنده است و آن را تحویل خواهد گرفت.

# الگوریتم توانمند (Bully) - ادامه

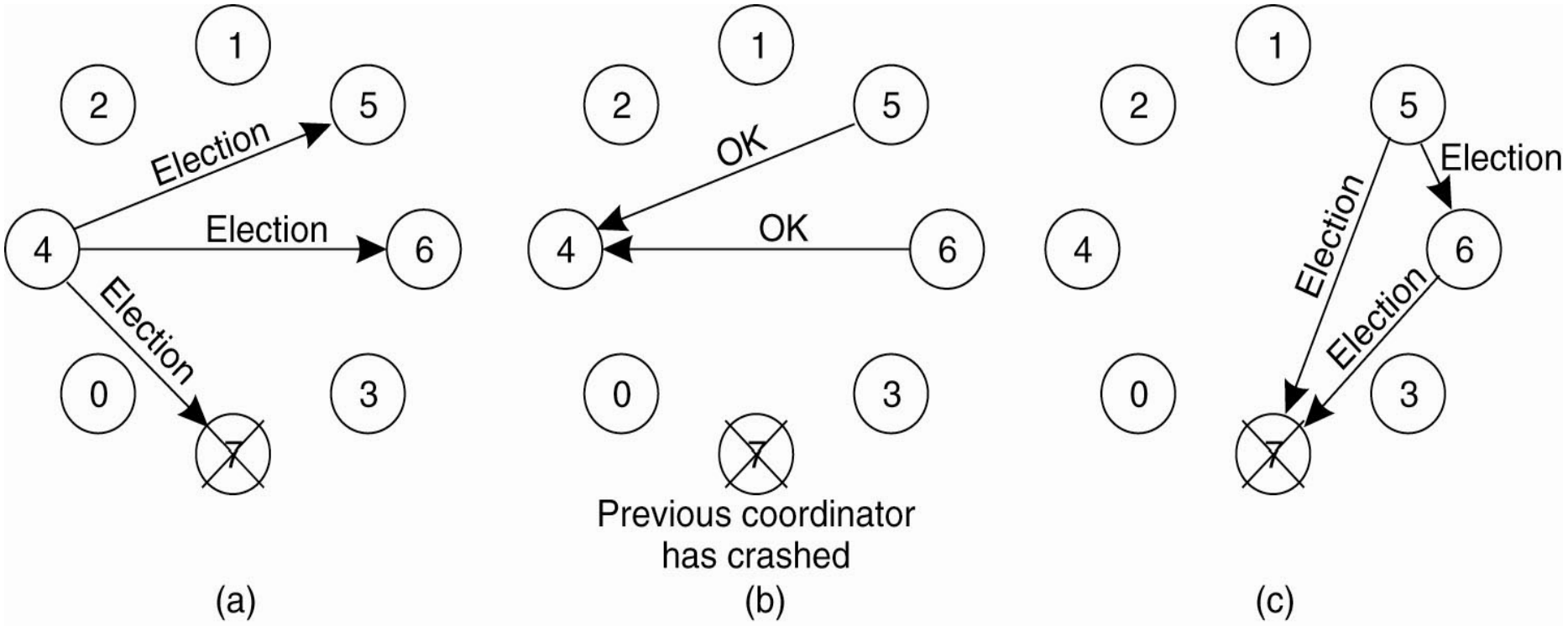
فرآیند P ، انتخاب را به صورت زیر انجام می دهد:

۱- P پیام ELECTION را به تمام فرآیندهایی با شماره بالاتر ارسال می کند.

۲- اگر هیچ کدام پاسخ ندهند، P انتخاب را برنده می شود و به عنوان هماهنگ کننده عمل می کند.

۳- اگر یکی از آن ها پاسخ دهد به عنوان هماهنگ کننده انتخاب می شود و کار P خاتمه می یابد.

# الگوریتم توانمند (Bully) – ادامه

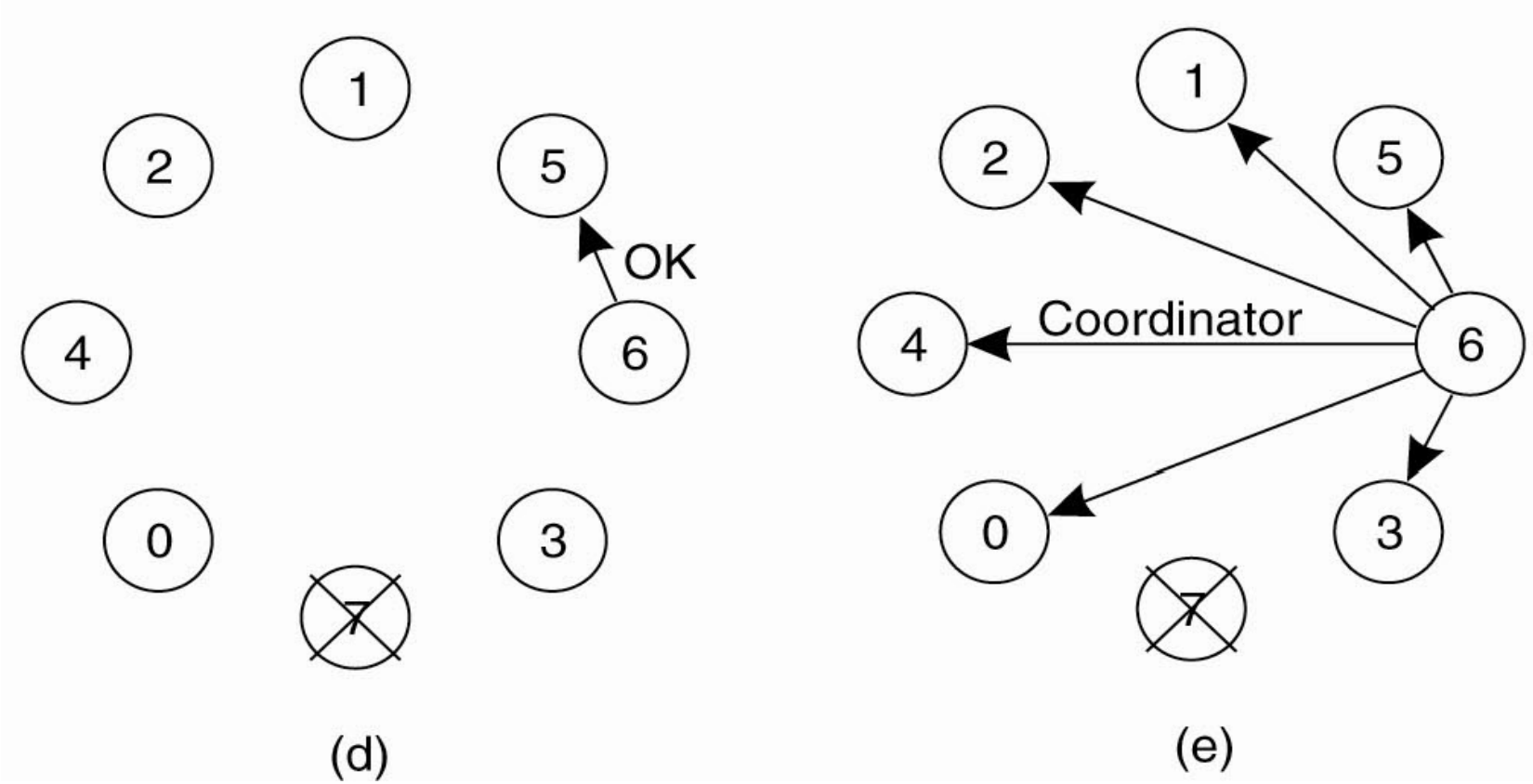



a) فرآیند ۴ انتخابات را انجام می دهد.

b) فرآیند های ۵ و ۶ پاسخ می دهند و به ۴ می گویند که متوقف شود.

c) اکنون ۵ و ۶ انتخابات را انجام می دهند.

# الگوریتم توانمند (Bully) – ادامه




d) فرآیندهای ۶ به ۵ می گوید که متوقف شود.

e) فرآیند ۶ برنده می شود و به همه خبر می دهد.

# الگوریتم حلقه (۱)

این الگوریتم از نشانه استفاده نمی کند.

فرض می کنیم فرآیندها به طور منطقی یا فیزیکی مرتب اند، به طوری که هر فرآیند می داند بعدی او کدام است.

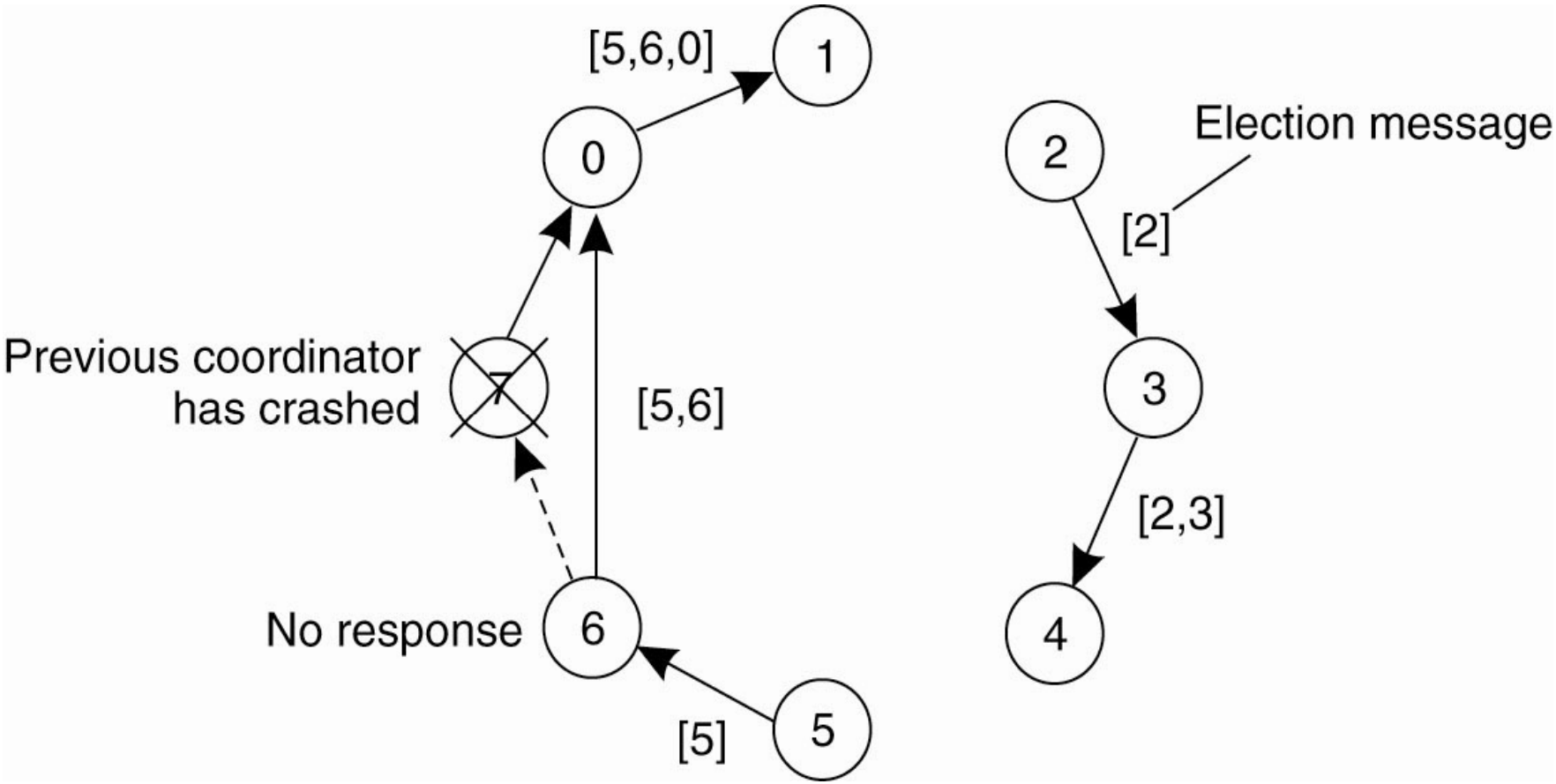
# الگوریتم حلقه (۲)

وقتی فرآیندی متوجه می شود که هماهنگ کننده عمل نمی کند، خودش را کاندیدای انتخاب هماهنگ کننده معرفی می کند یک پیام ELECTION می سازد که شامل شماره فرآیند خودش است و آن را به بعدی خودش می فرستد.

اگر بعدی او از بین برود، فرستنده از آن عبور می کند و به عدد بعدی در حلقه یا عدد بعد از خودش می رود تا یک فرآیند در حال اجرا پیدا شود.

سرانجام پیام به فرآیندی بر می گردد که آن را شروع کرده است و فرایند یک پیام COORDINATOR با بیشترین شماره برای ارسال به اعضای حلقه تولید میکند.

# الگوریتم حلقه (۳)




الگوریتم انتخاب با استفاده از حلقه. گرههای ۲ و ۵ متوجه خرابی هماهنگ کننده شده اند و

هر دو شروع به ارسال پیام کرده اند