Оперативни системи 2

Факултет техничких наука Косовска Митровица

Драгиша Миљковић

Питања на која ћемо одговорити

- Како ОС интерагује са И/О уређајима (тј. како проверава њихов статус, како шаље податке и контролне информације)?
- Шта је то драјвер (*driver*) уређаја?
- Које су компоненте драјвера хард-диска?
- Како се рачуна секвенцијална и насумична пропусност диска? (Тј. брзина уписивања и читања података)
- Који алгоритми се користе за распоређивање I/O захтева?

Улазно/излазни уређаји

I/O уређаји

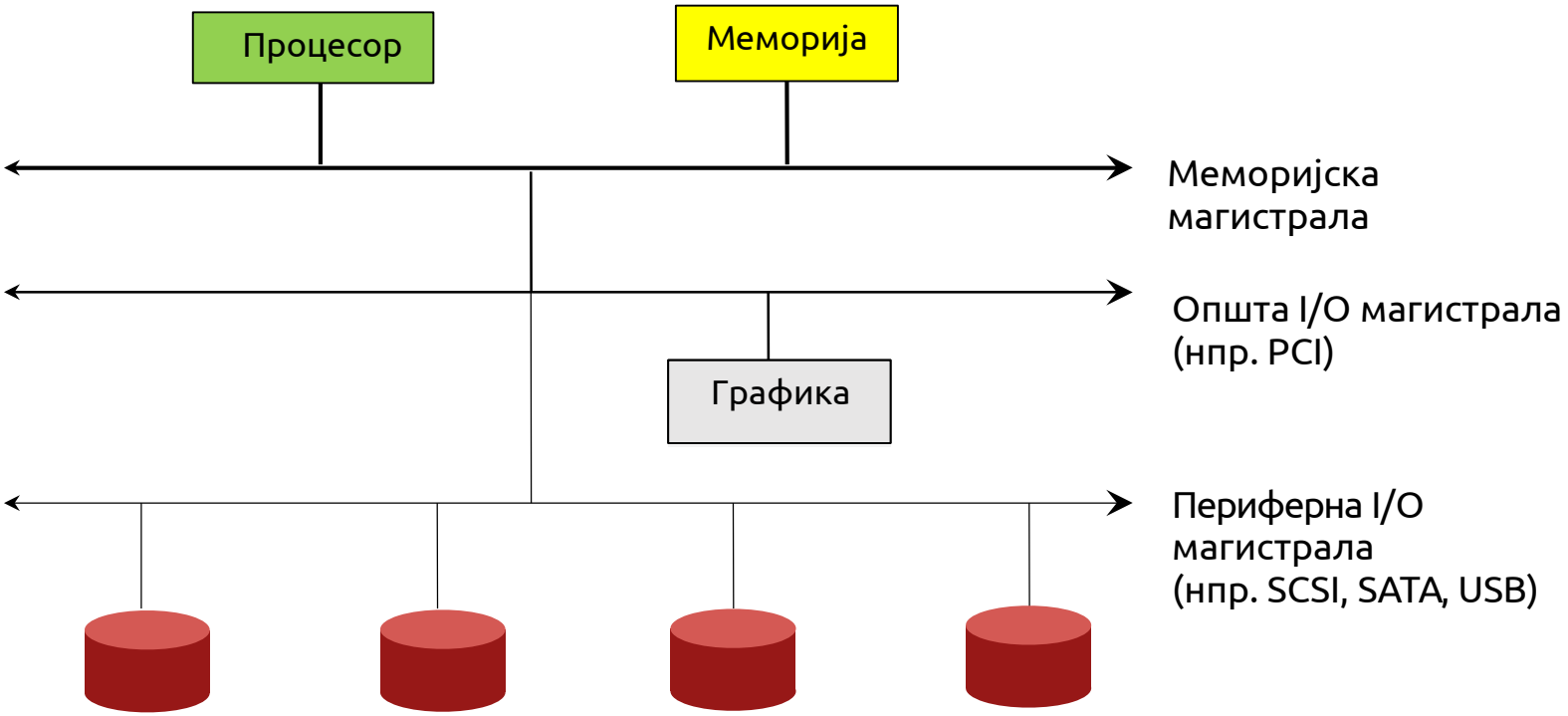
I/O је од круцијалне важности за омогућавање рачунарском систему да интерагује са системима.

- Шта бисте радили без тастатуре, монитора, дискова?

Проблем:

- Како би требало интегрисати I/O у оквиру система?
- Који су општи механизми?
- Како се то може ефикасно учинити?

Структура улазно/излазног (I/O) уређаја



Прототип системске архитектуре

Процесор је прикачен на главну меморију система преко неке врсте меморијске магистрале. Неки уређаји су повезани на систем преко опште I/O магистрале.

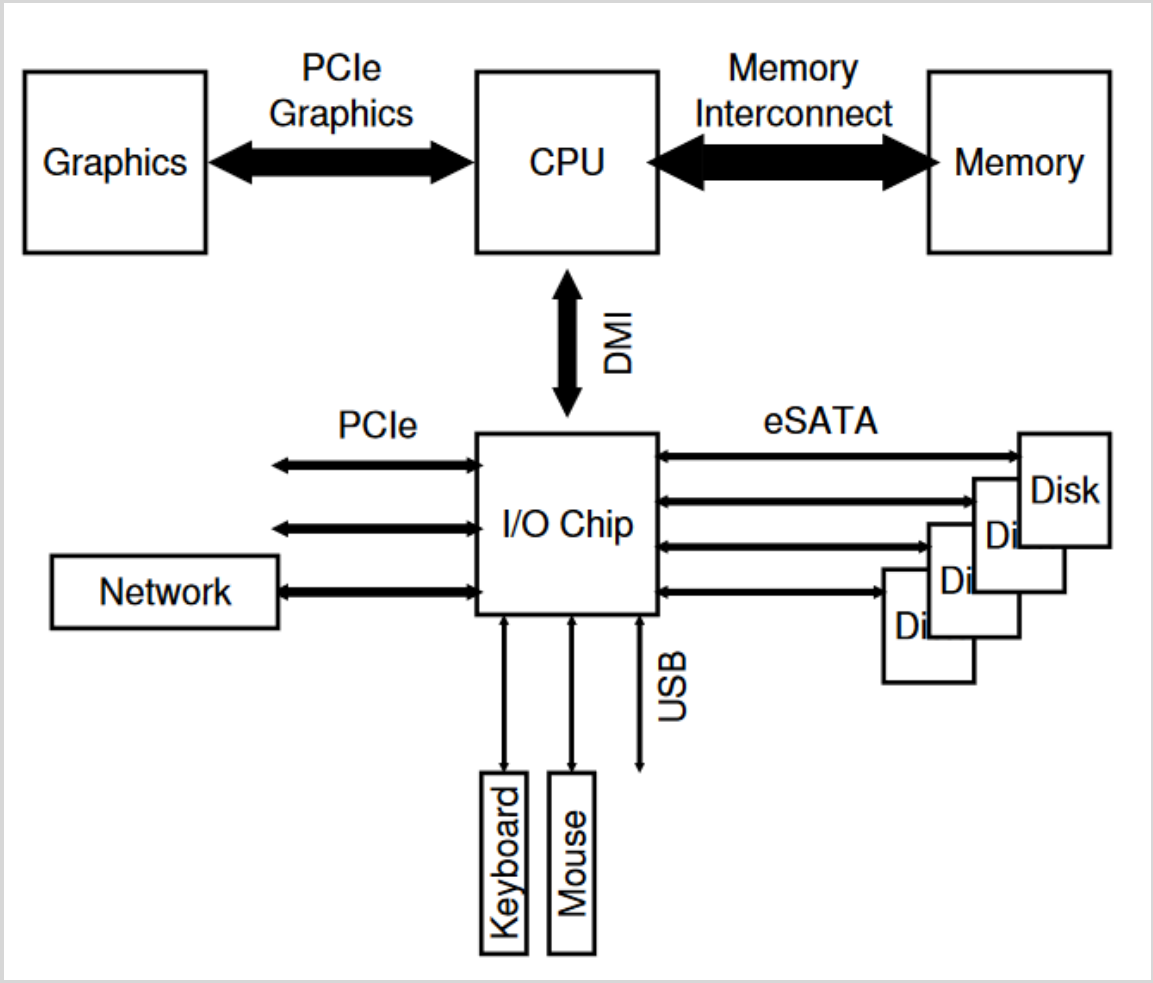
Архитектура I/O

Магистрале – стазе података које омогућавају размену података између процесора, RAM-а и I/O уређаја.

I/O магистрала – стаза података која повезује процесор и I/O уређај.

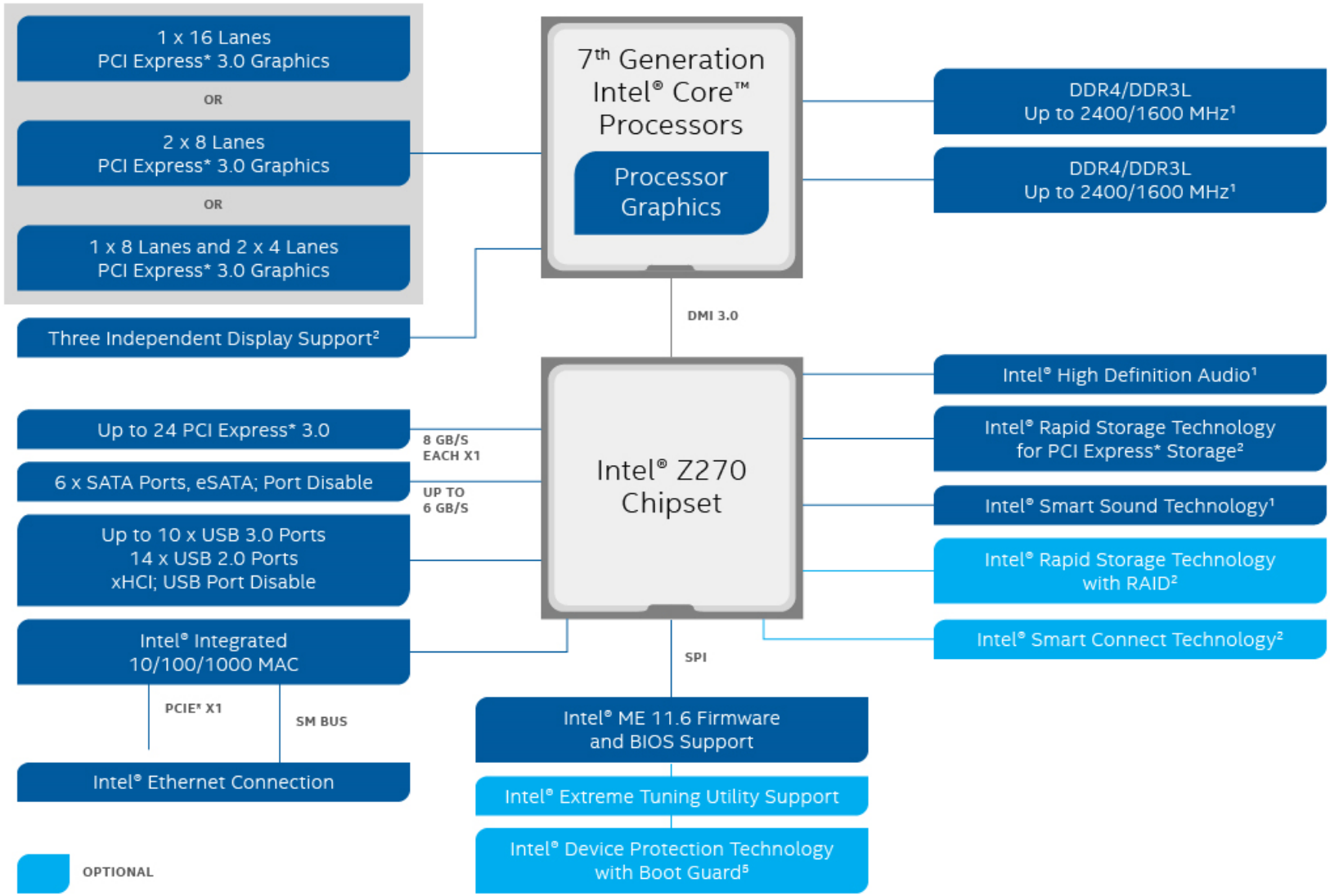
- I/O магистрала је повезана са I/O уређајем помоћу три врсти хардверских компоненти: I/O портова, интерфејса и контролера уређаја.

Пример модерне архитектуре – Intel Z270 Chipset



7th Gen Intel® Core™
Представљен у првом кварталу 2017. године.

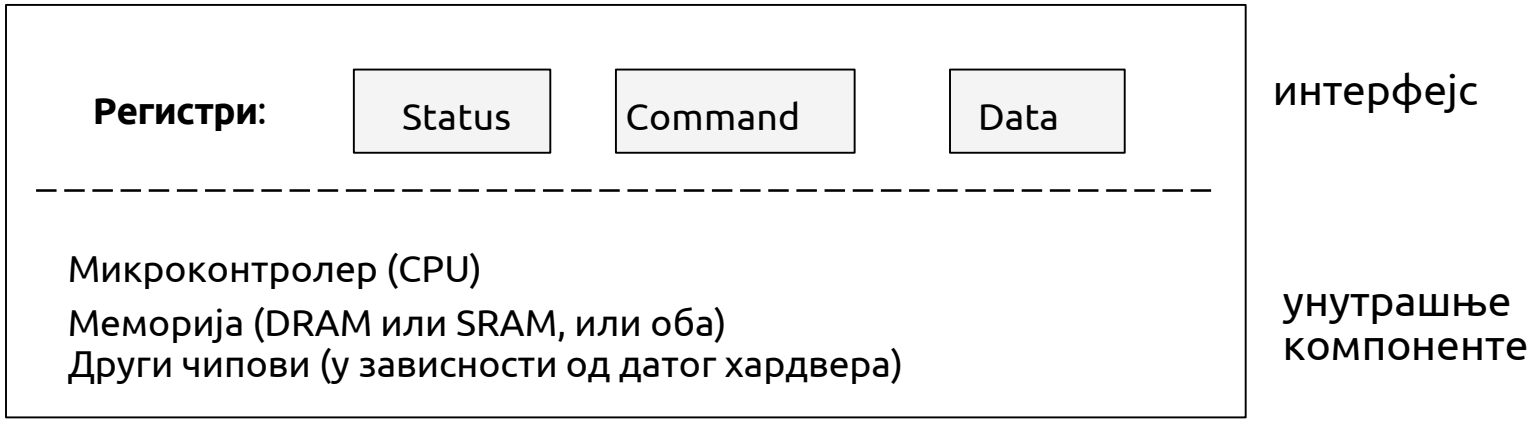
INTEL® Z270 CHIPSET BLOCK DIAGRAM



Основни модел уређаја

Општи модел уређаја (***Canonical Device***) има две важне компоненте:

- **Хардверски интерфејс** који је видљив споља и допушта системском софтверу да контролише његове операције. Њега користи ОС за читање и уписивање.
- **Унутрашње компоненте** које су сакривене и разликују се од имплементације до имплементације.



Основни модел уређаја

Хардверски интерфејс основног уређаја

Статусни регистар – чува тренутни статус уређаја.

Регистар наредбе – говори уређају да изврши одређен задатак.

Регистар података – прослеђује податке уређају или узима податке од уређаја.

Читањем и уписивањем у ова **три регистра**, оперативни систем може да **контролише понашање уређаја**.

Хардверски интерфејс основног уређаја (наставак)

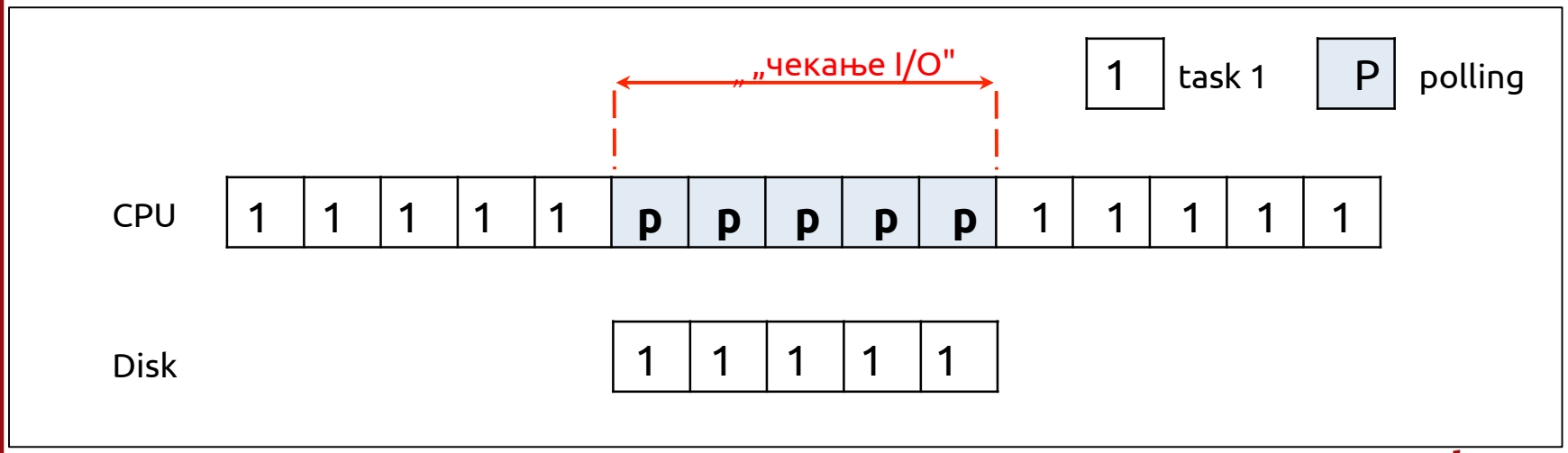
Типичан пример интеракције:

```
while (STATUS == BUSY)
    ; //vrti se i čeka da se uređaj oslobodi
Upis podataka u data registar
Upis komande u command registar
    // Ovo pokreće uređaj i izvršava komandu
while (STATUS == BUSY)
    ; // vrti se i čeka da uređaj završi sa zahtevom
```

Испитивање (*Polling*)

Оперативни систем изнова проверава статусни регистар док чека да уређај буде спреман.

- Позитиван аспект овога је то што је ово једноставно и ради како треба.
- Лоша страна овога је то што ово троши временске циклусе.
  - Пребацивање другог процеса на процесор док се чека омогућује бољу искоришћеност процесора.



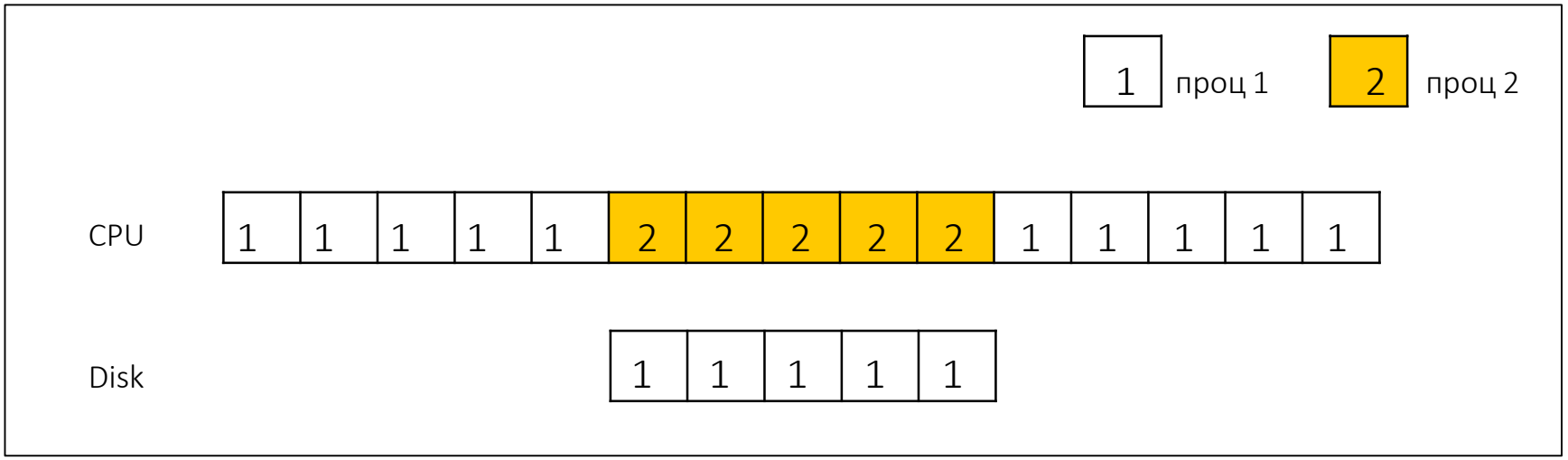
Дијаграм искоришћености процесора код испитивања

Прекиди

Процес који је захтевао I/O се успављује, а замењује се контекст новим процесом.

Када уређај заврши I/O, процес који чека на I/O се буди помоћу прекида.

- Позитиван аспект овога је то што се омогућује добра искоришћеност процесора и диска.



Дијаграм искоришћености процесора код прекида

Пример протокола уписивања


```
while (STATUS == BUSY)
     ; // spin
Upis podataka u DATA registar
Upis komande u COMMAND registar
while (STATUS == BUSY)
     ; // spin
```

Пример протокола уписивања

CPU:

Disk:

```
while (STATUS == BUSY)                  // 1
        ;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        ;
```

Пример протокола уписивања

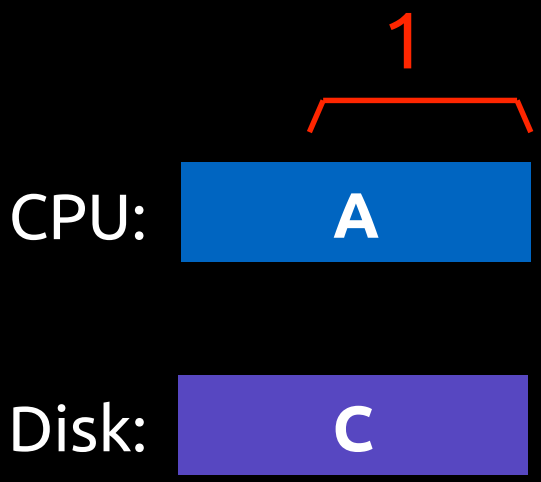
А жели да обави I/O

CPU: **A**

Disk: **C**

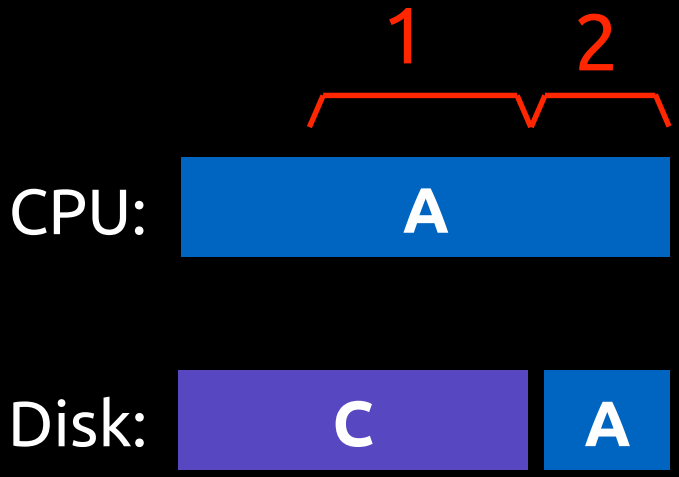
```
while (STATUS == BUSY)                  // 1
        ;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        ;
```

Пример протокола уписивања



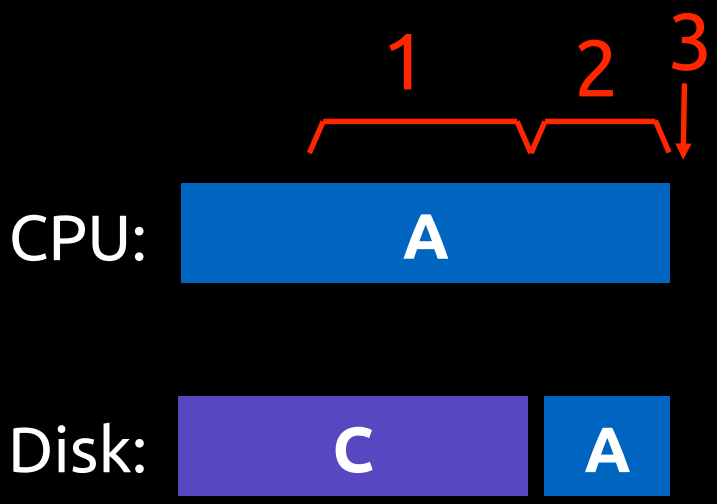

```
while (STATUS == BUSY)                  // 1
        ;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        ;
```

Пример протокола уписивања



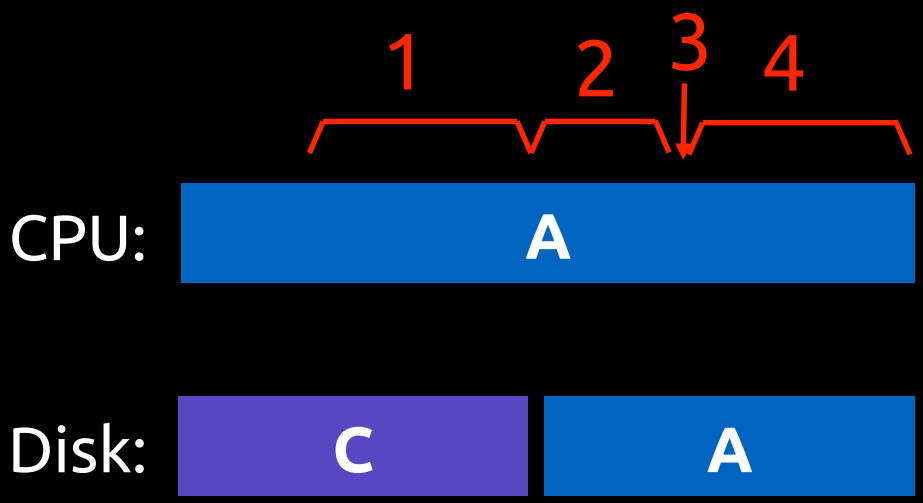

```
while (STATUS == BUSY)                  // 1
        ;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        ;
```

Пример протокола уписивања



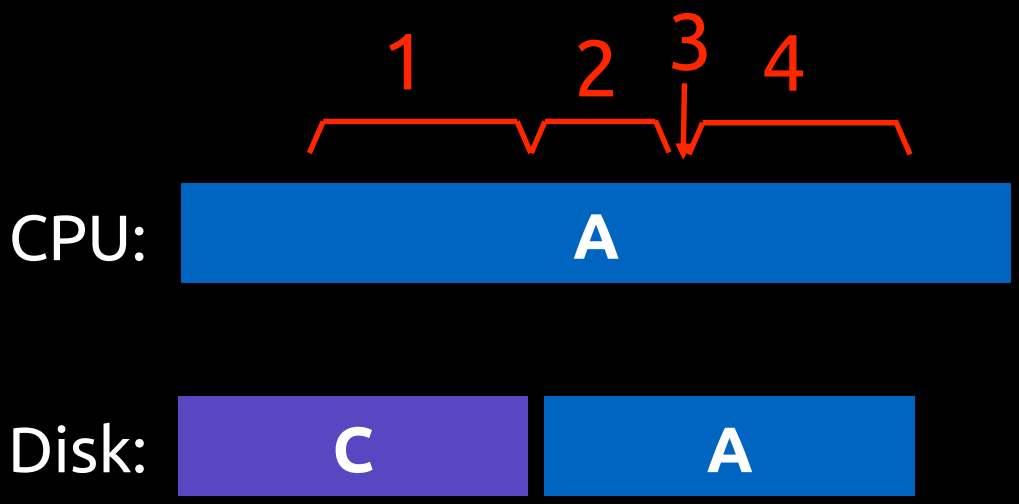

```
while (STATUS == BUSY)                  // 1
        ;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        ;
```

Пример протокола уписивања



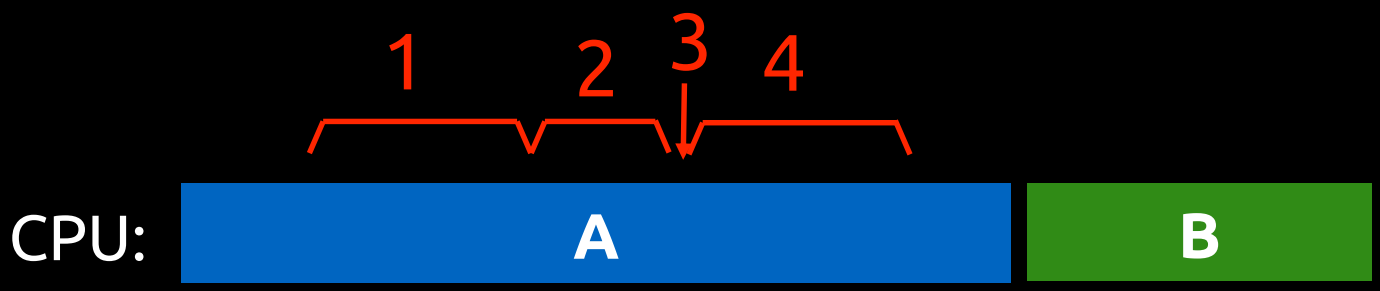

```
while (STATUS == BUSY)                  // 1
        ;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        ;
```

Пример протокола уписивања




```
while (STATUS == BUSY)                  // 1
        ;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        ;
```

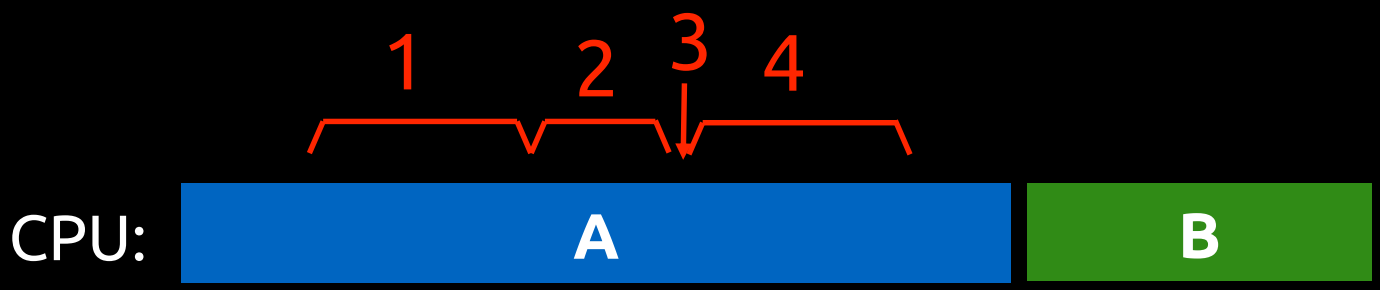
Пример протокола уписивања



Како избећи busy-waiting?
Помоћу прекида!

```
while (STATUS == BUSY)                  // 1
        ;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        ;
```

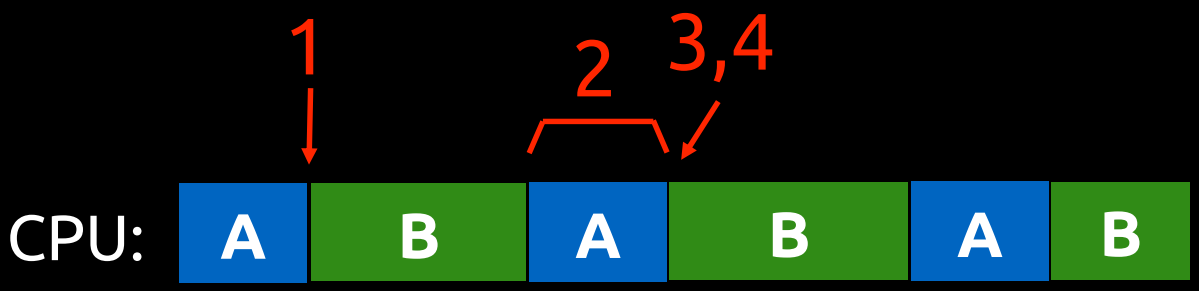
Пример протокола уписивања



Како избећи busy-waiting?
Помоћу прекида!

```
while (STATUS == BUSY)                    // 1
        čekanje na prekid;
Upis podataka u DATA registar             // 2
Upis komande u COMMAND registar           // 3
while (STATUS == BUSY)                    // 4
        čekanje na prekid;
```

Пример протокола уписивања



Како избећи busy-waiting?
Помоћу прекида!

```
while (STATUS == BUSY)                    // 1
        čekanje na prekid;
Upis podataka u DATA registar             // 2
Upis komande u COMMAND registar           // 3
while (STATUS == BUSY)                    // 4
        čekanje na prekid;
```

Испитивање или прекиди?

Ипак, прекиди нису у свим случајевима најбоље решење.

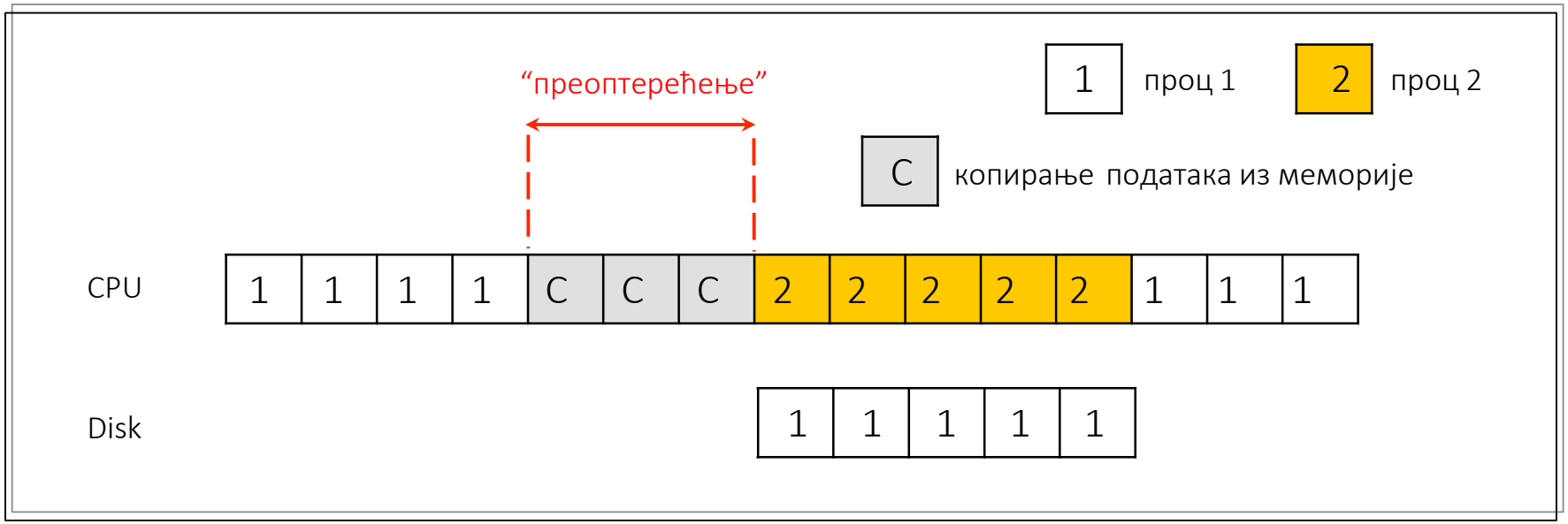
- Уколико уређај има веома брзе перформансе, тада ће прекиди заправо успорити систем (јер је замена контекста „скупља“ у односу на чекање).

> Уколико је уређај брз – боље је испитивање.
>
> Уколико је уређај спор – бољи су прекиди.

- Шта радити када су непознате перформансе уређаја? Хибридни приступ – прво испитивање, па прекиди.
- Може се десити да дође до „поплаве“ прекида:
  - Тада може доћи до живе петље (јер се све време губи на руковање прекидима). У тој ситуацији је боље неко време игнорисати прекиде док се неки опслужују.
- Још једно побољшање – груписање прекида (спајање неколико прекида).

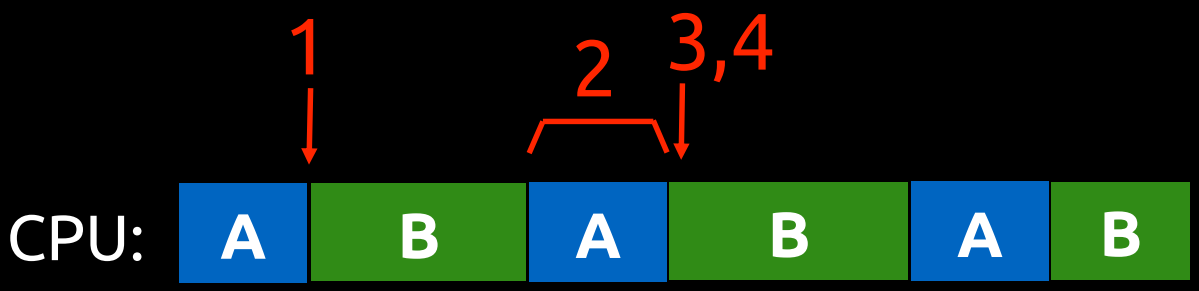
Прцесор је и даље преоптерећен

Процесор трађи много времена и копирајући податке из меморије (*реч по реч*).



Дијаграм искоришћености процесора

Пример протокола уписивања



Шта још може да се оптимизује?

ПРЕНОС ПОДАТАКА!

```
while (STATUS == BUSY)                  // 1
        čekanje na prekid;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        čekanje na prekid;
```

Programmed I/O vs. Direct Memory Access

Програмирани I/O је приступ код кога CPU говори уређају шта су подаци

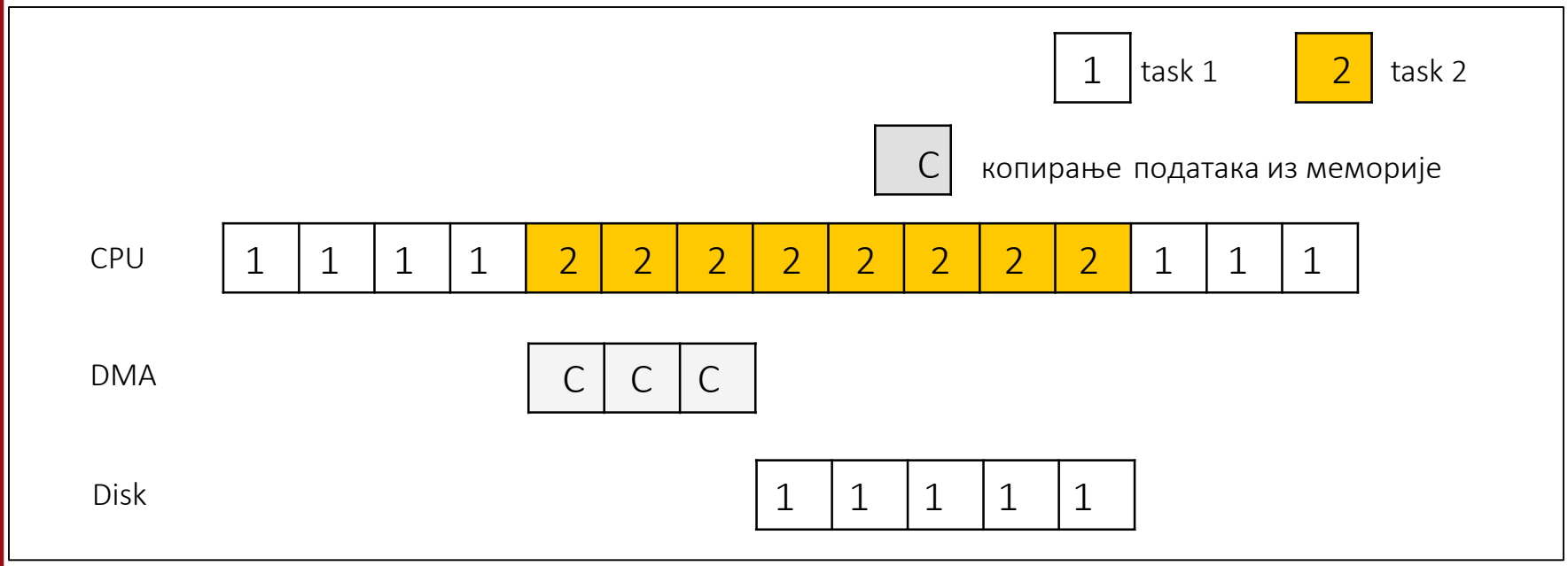
Други приступ – DMA:

- CPU само остави податке у меморији, а уређај директно чита податке из ње.

DMA (Direct Memory Access)

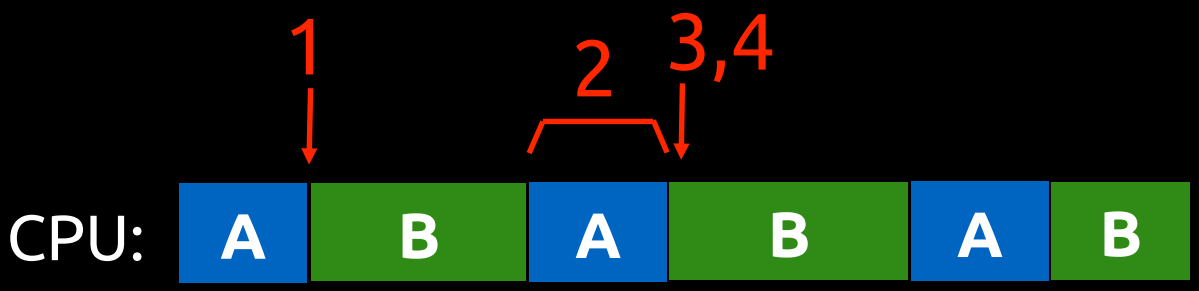
Копира податке знајући „где се у меморији подаци налазе и колико података треба копирати“.

Када заврши копирање, DMA подиже прекид, и тада почиње I/O на диску.



Дијаграм искоришћености процесора преко DMA

Пример протокола уписивања



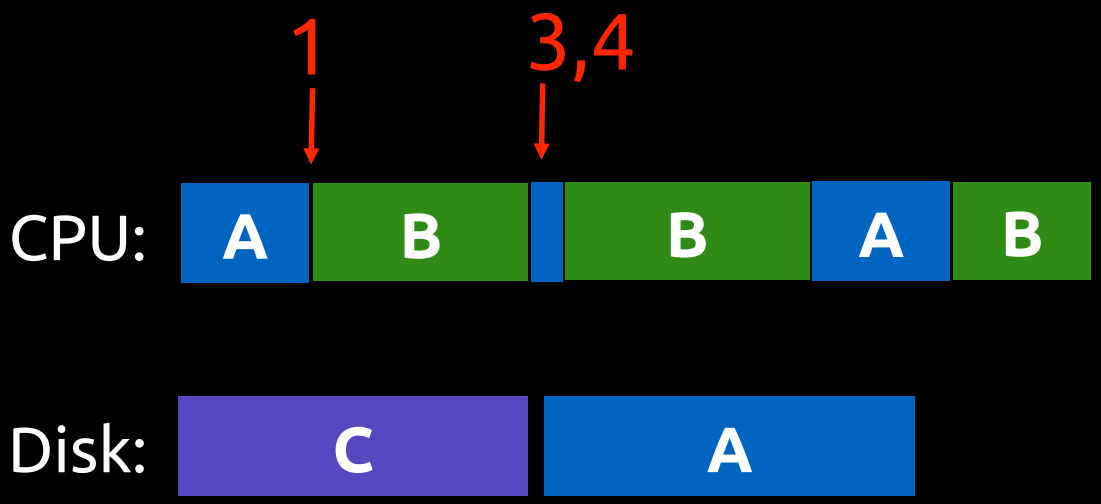
Шта још може да се оптимизује?

Пренос података!

Избацујемо уписивање у DATA регистар.

```
while (STATUS == BUSY)                  // 1
        čekanje na prekid;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        čekanje na prekid;
```

Пример протокола уписивања




```
while (STATUS == BUSY)                  // 1
        čekanje na prekid;
Upis podataka u DATA registar           // 2
Upis komande u COMMAND registar         // 3
while (STATUS == BUSY)                  // 4
        čekanje na prekid;
```

Нема више чекања

Интеракција са уређајем (контрола)

Како оперативни систем интерагује са уређајем?

Могућа решења (оба се користе у модерним системима):

- **I/O инструкције**: ово представља начин да ОС (и само он!) пошаље податке одређеним регистрима уређаја (сваки уређај има свој порт).
  - Нпр. *in* и *out* инструкције код x86.

- **меморијски-мапиран I/O**
  - Регистри уређаја су доступни као да су то меморијске локације. Хардвер их мапира на адресни простор.
  - Оперативни систем уписује (*load*) и чува (*store*) податке директно на уређај (уместо у главну меморију).

Интеракција са уређајем (наставак)

Али како оперативни систем интерагује са **различитим** интерфејсима?

- Нпр. ако се жели направити систем датотека који ради са SCSI дисковима, IDE дисковима, USB драјверима, и тако даље.

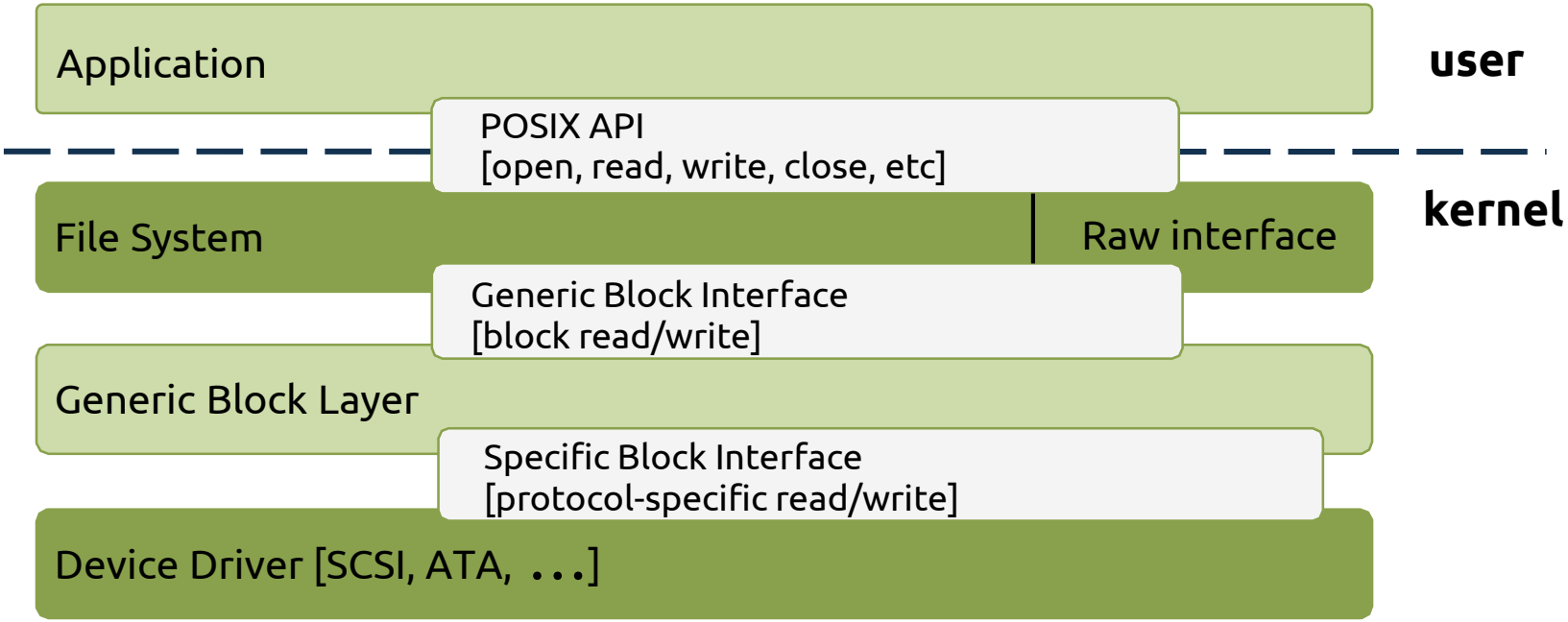
Решење: **апстракција**

- Апстракција енкапсулира информације о интеракцији са уређајем.

Апстраховање система датотека

Испод је дат пример апстракције коју Линукс користи за имплементацију система датотека.

- Систем датотека само **назначује** који диск користи, не зна ништа о имплементацији. Нпр. он захтеве за читање или уписивање издаје генеричком слоју.



Стек система датотека

Проблем код апстраховања система датотека

Уколико постоји уређај који има неке посебне способности, ове способности ће бити неискоришћене на слоју генеричког интерфејса.

Постоји много, много различитих уређаја, сваки од њих има неки свој протокол. Како да се избегне писање различитог ОС за сваку могућу хардверску комбинацију?

- Писањем драјвера за сваки од уређаја.

Преко 70% програмског кода оперативног система се налази у драјверима уређаја.

- Сви драјвери уређаја су потребни јер ће их корисник можда прикључити на систем.
- Они су и главни узрок проблема за кернел, и доводе до разних грешака.

Апстраховање уређаја за смештање података

апликације

систем датотека

распоређивач

драјвер

хард-диск

Потребно је изградити заједнички интерфејс изнад свих хард-дискова

Неке од вредности кашњења чувања података које би требало да знате

https://dzone.com/articles/every-programmer-should-know

https://dev.to/sahilrajput/how-latency-numbers-changes-from-1990-to-2020-173n

Хард-диск

Хард-диск

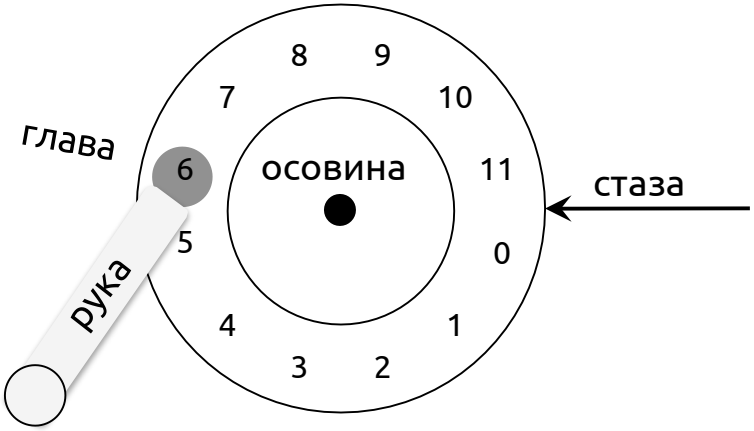
Диск чини адресни простор који се састоји из великог броја адресабилних сектора.

- Сектор – обично је њихова величина 512 бајтова или 4096 бајтова.
- Главне операције у раду са хард-диском су читање и уписивање сектора.

Сектори диска су нумерисани бројевима од 0 до n-1.

Произвођач уређаја гарантује да ће **уписивање на ма који сектор** (512 бајтова) бити **атомично**.

Основна структура диска



Диск са једном стазом

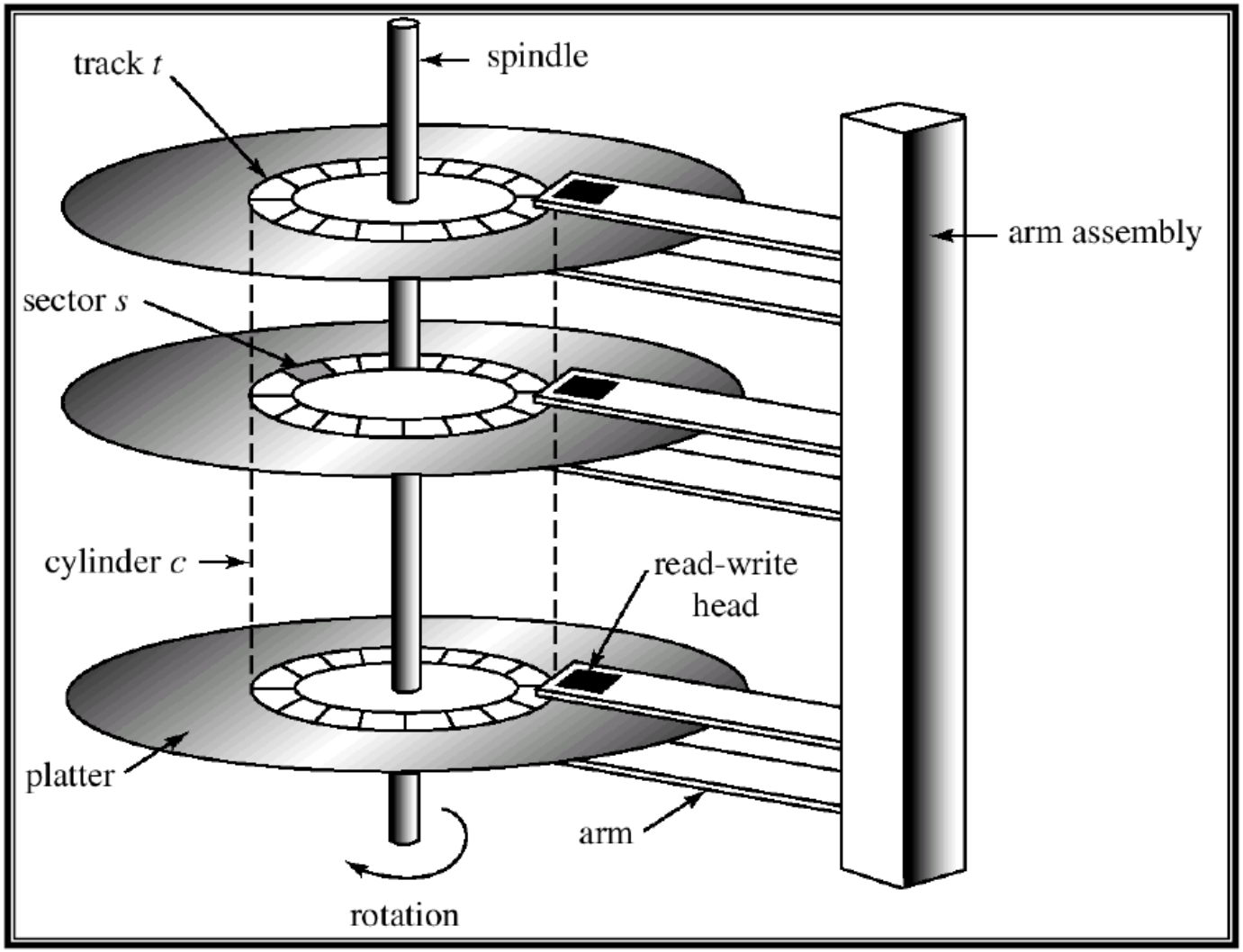
Плоче: кружне тврде површине на које се подаци уписују. Прекривене су магнетним филмом.

- Плоче су наслагане око **осовине**, око које се и окрећу.
- Плоча садржи на хиљаде **стаза**.

Главе диска: читају и уписују податке са сваке стазе.

- Глава диска је прикачена на једну **руку диска**, а она се помера по површини да би позиционирала главу на жељену стазу.

Основна структура диска (наставак)

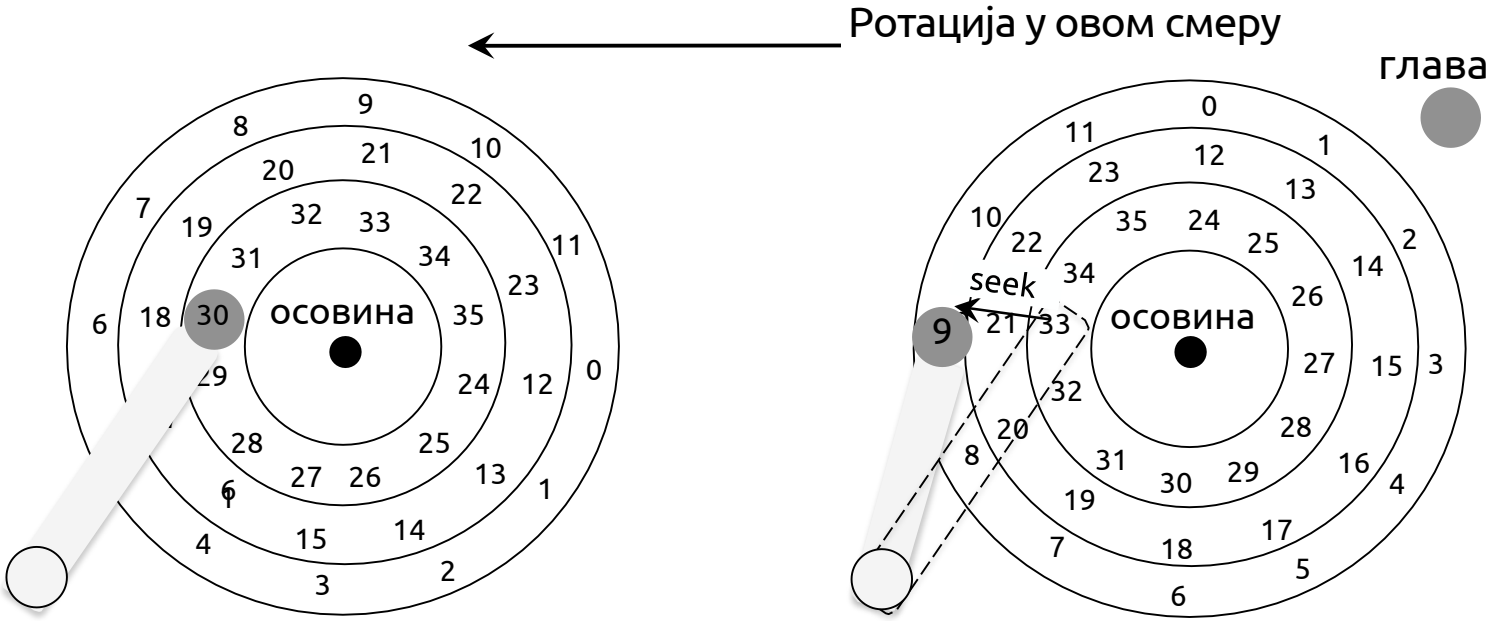


Демонстрација рада

https://www.youtube.com/watch?v=9eMWG3fwiEU&feature=youtu.be&t=30s

https://www.youtube.com/watch?v=L0nbo1VOF4M

Време претраге

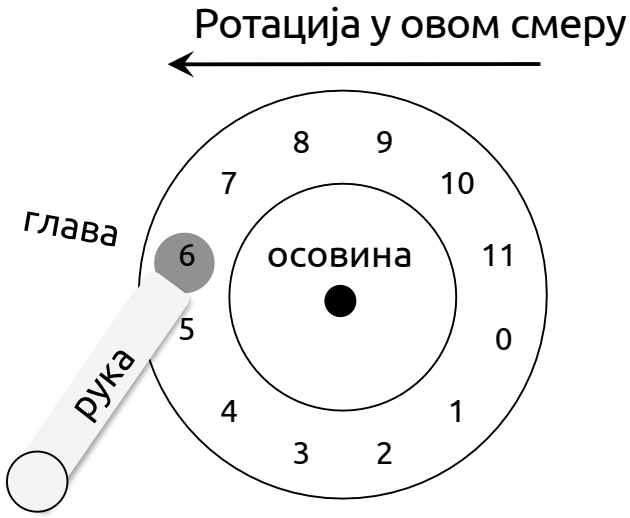


Диск са три стазе (десно – пребацивање на сектор 9)

Време позиционирања (seek time): Ово је време које је потребно да би се глава померила на стазу која садржи жељени сектор.

- Рука диска се помера до жељене стазе. (Плоча се, наравно, ротира.)
- Постоји додатни трошак кашњења услед ротације.

Ротационо кашњење



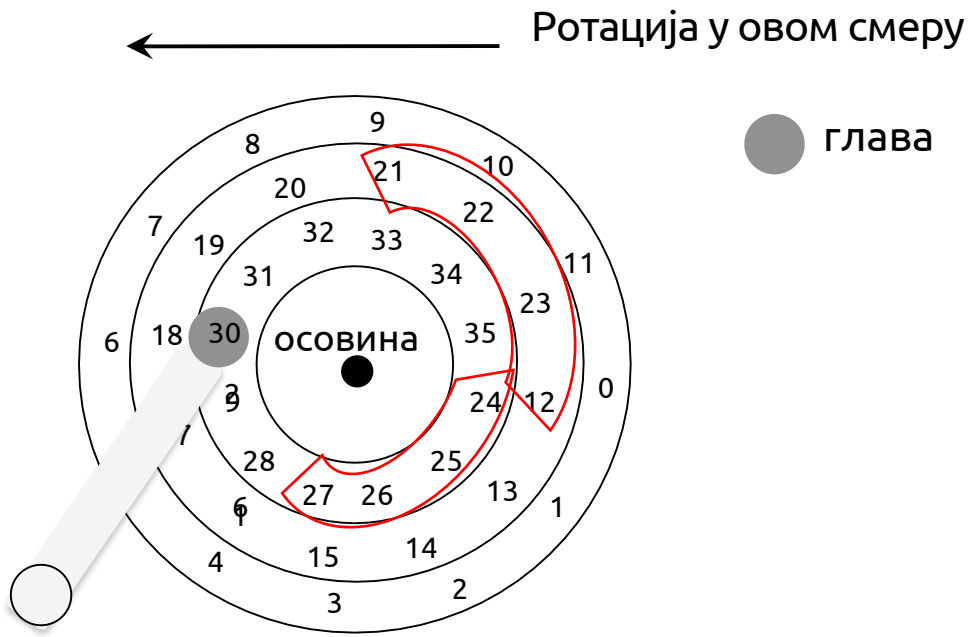
Диск са једном стазом

Ротационо кашњење: Време које је потребно да се жељени сектор ротира.

- Нпр. пуно ротационо кашњење је R.
  - Читање сектора 0: ротационо кашњење = R/2
  - Читање сектора 5: ротационо кашњење = R – 1 (у најгорем случају)

Искошене стазе (*Track skew*)

Многи хард-дискови покушавају да оптимизују секвенцијална читања у случајевима када треба померати главу у суседне траке. Овако се постиже да се код померања главе она одмах смести на одговарајући сектор и минимизује се ротационо кашњење.



Диск са три стазе

Кеш (бафер стаза)

Ово је важан део свих модерних хард-дискова. Он је мала и брза меморија (обично 8 до 16 мегабајта) коју уређај користи да чува податке који се уписују или читају са диска.

Нпр. Када се чита неки сектор са диска, уређај може да одлучи да прочита све секторе те стазе и кешира их, тако да ће моћи да брзо одговори уколико стигне захтев за читање неког од тих сектора.

Write back / Write through

Код уписивања на диск, уређај може да бира, да ли да јави да је уписивање завршено када смести податке у меморију, или када те податке копира на сам диск. Ово прво се назива ***write back*** кеширање (неки га називају и тренутно извештавање – *immediate reporting*), а други приступ се назива ***write through***.

Први приступ чини да уређај одаје утисак да је „бржи“, али је опасан; уколико систем датотека или апликације захтевају да се подаци уписују по одређеном редоследу, ово кеширање може да доведе до проблема.

Распоређивач диска

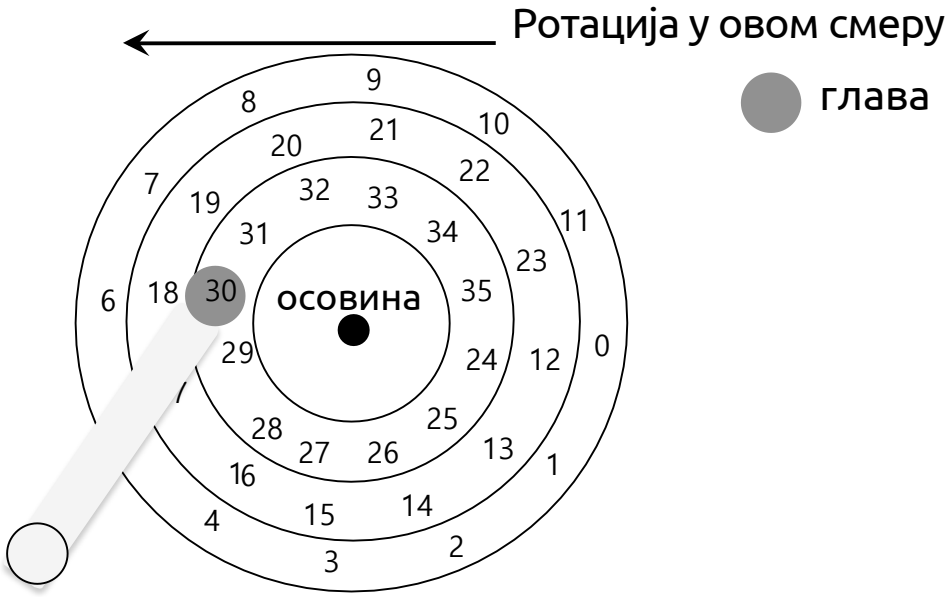
Услед тога што I/O троши много времена, ОС мора да одлучује о распоређивању I/O захтева који се упућују диску, тј. мора да проучи захтеве и донесе одлуку који следећи да опслужи.

За разлику од распоређивања процеса где је трајање обично непознато, овде се може дати добра процена колико ће неки захтев да троши времена.

Проучавањем времена претраге и могућег ротационог кашњења, распоређивач зна колико времена треба за неки захтев и може донети одлуку кога да распореди следећег.

Обично се ОС труди да примени принципе **SJF** (*shortest job first*) рапоређивача.

Распоређивач диска - наставак



SSTF – ***Shortest Seek Time First*** – код овог приступа се бира захтев који ће најбрже бити опслужен. (Ово није фер јер стазе у средини имају најбоље шансе да буду опслужене.

Лифт (***Elevator***) илити **SCAN** – код овог приступа се захтеви опслужују прво у једном, па у другом правцу (од крајње унутрашње стазе ка спољашњим, и обрнуто). И овде унутрашње стазе имају дупло више шансе да буду одабране, па постоји варијанта **C-SCAN** (***Circular*** **SCAN**) у којој се претрага врши само у једном правцу, нпр. увек се креће од унутрашње стазе и иде ка спољашњим, и тако укруг.

Распоређивач диска - наставак

SPTF – ***Shortest Positioning Time First*** – овде се води рачуна о односу времена претраге и ротационог кашњења, тако да се када је време претраге много веће од ротационог кашњења користи SSTF. У обрнутом случају треба одабрати позиционирање на стазу која је даља.

Ово је услед тога што су на модерним уређајима ротационо кашњење и време претраге отприлике исти (наравно у зависности од појединачних захтева). Тако да SPTF побољшава перформансе.

SPTF је тешко имплементирати на нивоу ОС, тако да се ово обично врши на унутар самог уређаја. Више речи о томе на следећем слајду.

Још нешто о распоређивању

Где се врши распоређивање операција на диску? На старијим системима је ово радио ОС, тада су и дискови били једноставнији.

На модерним системима дискови могу и да опслуже више захтева истовремено и имају комплексне интерне распоређиваче (који могу исправно имплементирати SPTF). Унутар контролера диска су доступне све потребне информације, укључујући и тачну позицију главе.

Сједињавање I/O захтева (I/O *merging*) – захтеви за уписивање на суседне блокове ће бити сједињени (тиме се смањује број захтева које диск опслужује).

Колико дуго треба ОС да чека пре него што проследи I/O захтев према диску?

- work-conserving – захтеви се одмах прослеђују диску
- work-nonconserving – чека се неки кратак период (логика иза овога је да ће можда наићи неки захтев који је „боље“ опслужити пре других, или се неки захтеви могу сјединити, итд.)

Симулатор распоређивања

http://imi.pmf.kg.ac.rs/~milos/simdisk/

Поређење перформанси дискова

	Cheetah	Barracuda
Капацитет	300 GB	1 TB
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s
Број плоча	4	4
Величина кеша	16 MB	32 MB

Секвенцијални послови – пропусност?
Cheetah: 125 MB/s.
Barracuda: 105 MB/s.

Поређење перформанси дискова

	Cheetah	Barracuda
Капацитет	300 GB	1 TB
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s
Број плоча	4	4
Величина кеша	16 MB	32 MB

Насумични послови – пропусност?
Да бисмо то одредили, претпоставићемо насумично читање величине 16KB.

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

Колико је просечно потребно времена да се прочита 16KB са *Cheetah* диска?

$$Време_{i/o} = seek + rotation + transfer$$

Seek = 4 ms

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

Колико је просечно потребно времена да се прочита 16KB са *Cheetah* диска?

Колико је просечно време ротације?

$$\text{просечна ротација} = \frac{1}{2} \times \frac{1\ \text{min}}{15000} \times \frac{60\ \text{sec}}{1\ \text{min}} \times \frac{1000\ \text{ms}}{1\ \text{sec}} = 2\ \text{ms}$$

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

Колико је просечно потребно времена да се прочита 16KB са *Cheetah* диска?

А да се пренесе 16KB?

$$\text{пренос} = \frac{1\ \text{sec}}{125\ \text{MB}} \times 16\ \text{KB} \times \frac{1.000.000\ \text{us}}{1\ \text{sec}} = 125\ \text{us}$$

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

Колико је просечно потребно времена да се прочита 16KB са *Cheetah* диска?

Време_Cheetah = 4ms + 2ms + 125us = 6,1ms

Колика је пропусност?
Величина преноса подељена са временом.

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

Колико је просечно потребно времена да се прочита 16KB са *Cheetah* диска?

Време_Cheetah = 4ms + 2ms + 125us = 6.1ms

$$\text{пропусност} = \frac{16\ \text{KB}}{6{,}1\text{ms}} \times \frac{1\ \text{MB}}{1024\ \text{KB}} \times \frac{1000\ \text{ms}}{1\ \text{sec}} = 2{,}5\ \text{MB/s}$$

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

Колико је просечно потребно времена да се прочита 16KB са *Barracuda* диска?

Време = seek + rotation + transfer

Seek = 9 ms

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

Колико је просечно потребно времена да се прочита 16KB са *Barracuda* диска?

Колико је просечно време ротације?

$$\text{просечна ротација} = \frac{1}{2} \times \frac{1\ \text{min}}{7200} \times \frac{60\ \text{sec}}{1\ \text{min}} \times \frac{1000\ \text{ms}}{1\ \text{sec}} = 4{,}1\ \text{ms}$$

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

Колико је просечно потребно времена да се прочита 16KB са *Barracuda* диска?

А да се пренесе 16KB?

$$\text{пренос} = \frac{1\ \text{sec}}{105\ \text{MB}} \times 16\ \text{KB} \times \frac{1.000.000\ \text{us}}{1\ \text{sec}} = 149\ \text{us}$$

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

Колико је просечно потребно времена да се прочита 16KB са *Barracuda* диска?

Време_Barracuda = 9ms + 4,1ms + 149us = 13,2ms

Колика је пропусност?

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

Колико је просечно потребно времена да се прочита 16KB са *Barracuda* диска?

Време_Barracuda = 9ms + 4,1ms + 149us = 13,2ms

$$\text{пропусност} = \frac{16\ \text{KB}}{13{,}2\text{ms}} \times \frac{1\ \text{MB}}{1024\ \text{KB}} \times \frac{1000\ \text{ms}}{1\ \text{sec}} = 1{,}2\ \text{MB/s}$$

Поређење перформанси дискова

	Cheetah	Barracuda
RPM	15.000	7.200
Време претраге	4 ms	9 ms
Макс. брзина преноса	125 MB/s	105 MB/s

	Cheetah	Barracuda
Секвенцијално	125 MB/s	105 MB/s
Насумично	2.5 MB/s	1.2 MB/s