Паралелно Програмиране

Въведение

доц. д-р Александър Пенев

Какво е Паралелно Програмиране?

Въпрос с Прост Отговор:

Използването на повече от един процесор/процес, за изпълнението на дадена задача.

и ...

Сложно Съдържание

Въведение

Теми разглеждани в курса

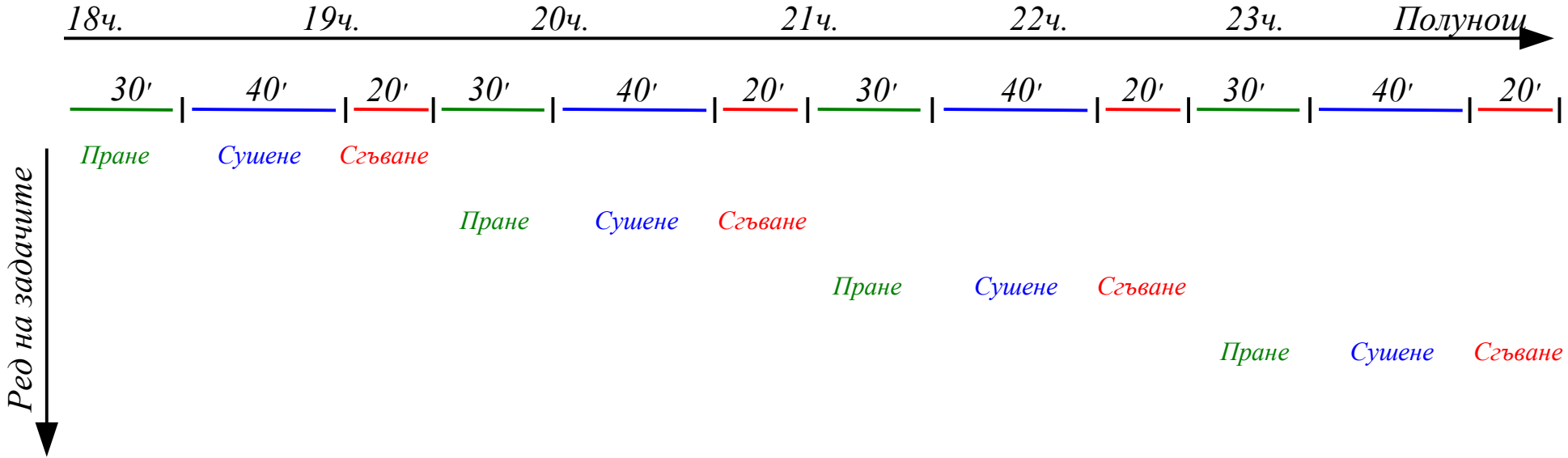
1. Въведение в паралелното програмиране.
   Пример. Кратка история. Базови понятия и концепции. Паралелни архитектури. Модели за паралелно програмиране. Възможни проблеми. Пример – критична секция. Практически проект.
2. Съвременни паралелни архитектури. Класификация на Флин (Flynn). CPU, GPGPU, HSA, Високо производителни компютри (HPC).
   Класификация на Flynn (SISD SIMD SIMT MISD MIMD). Скаларни/Pipelined и Суперскаларни процесори. SIMD инструкции. Dataflow architecture. Vector processor. Multiprocessor (symmetric/asymmetric). CPU/GPGPU. HSA. HPC.
3. Памет. Йерархия на паметта. Архитектури на паметта на паралелните компютри.
   Памет. Кеш. Архитектури (shared, distributed, distributed shared, UMA, NUMA, COMA), Shared-Nothing Architecture и др. Massively parallel computer (GRID Системи, Компютърни Клъстъри, Силно паралелни процесорни масиви).
4. Видове паралелизъм. Векторизация. Задачи – фибри, нишки, процеси. Видове многозадачност.
   Видове паралелизъм: Bit Level; Instruction (ILP); Task; Data; Memory. Векторизация. Векторизация на цикли. Задачи – фибри, нишки, процеси. Видове многозадачност: Temporal; Simultaneous (SMT); Speculative (SpMT); Preemptive; Cooperative; Clustered Multi-Thread (CMT).
5. Теоретични аспекти на паралелните алгоритми. Анализ на паралелни алгоритми. Зависимости на данните, структурата и контрола (Data dependency, Data, Structural and Control Hazards).
   PRAM модел. PEM модел. Разпаралеляване. Анализ на паралелни алгоритми. Критичен път. Закони на Amdahl. Закон на Gustafson. Метрики на Karp–Flatt. Забавяне и Ускорение. Коефициенти и метрики. Granularity...

Теми разглеждани в курса

6. Класически алгоритми и проблеми. Алгоритъм на Декер. Проблеми при паралелните алгоритми: Мъртва хватка, жива хватка, трудна скалируемост, глад за ресурси, съперничество и др. Producer-Consumer.
   Задача за „Вечерящите философи“ и др. Алгоритъм на Декер. Задача за „Тримата пушачи“. Задача за „Спящият бръснар“. Deadlock, Livelock, Parallel slowdown, Race condition, Software lockout, Scalability, Starvation, Convoying, Contention; Deterministic algorithms; Embarrassingly parallel; Producer-Consumer.
7. Модели за паралелно програмиране. Координация в паралелните алгоритми.
   Модели за ПП: Shared Memory model; Threads model; Message Passing model; Implicit interaction model; Data Parallel model. Видове координация/синхронизация в паралелните алгоритми: Barrier; Locks; Semaphores; Mutexes, ...
8. Дизайн на паралелни програми.
   Проблем и решения. Декомпозиция на данни и алгоритми. Видове комуникация между подзадачите. В/И – проблеми и решения. Fork-Join. Map-Reduce. Hotspots и Bottlenecks. Не блокиращи алгоритми и структури данни. Zero-Copy. Read-Copy-Update (RCU/COW). Транзакционна памет.
9. Езици и Библиотеки (API) за паралелно програмиране.
   Езици за ПП. Примери. Библиотеки: C++11 STL (Futures, Promises, Threads, ...); C# (Threads, TPL, Tasks, Furures, PLINQ, async, await, yield, ...); Java (Threas, Locks, Atomics, Futures, Streams, ...); OpenMP (Fork-Join, ...); MPI/MPI-2 (Комуникация, синхронизация, паралелен В/И, редукции, ...); POSIX Threads, Boost.Thread, TBB; CUDA (Примери, PyCUDA, ...); OpenCL (); OpenHMPP, OpenACC, C++ AMP (Коделети, кернели, #pragma, GPU, ...).
10. Бъдеще на паралелните архитектури и програмиране.
    GPGPU, TPU, FPGA, ASIC, NPU, QPU – Квантов паралелизъм и др.

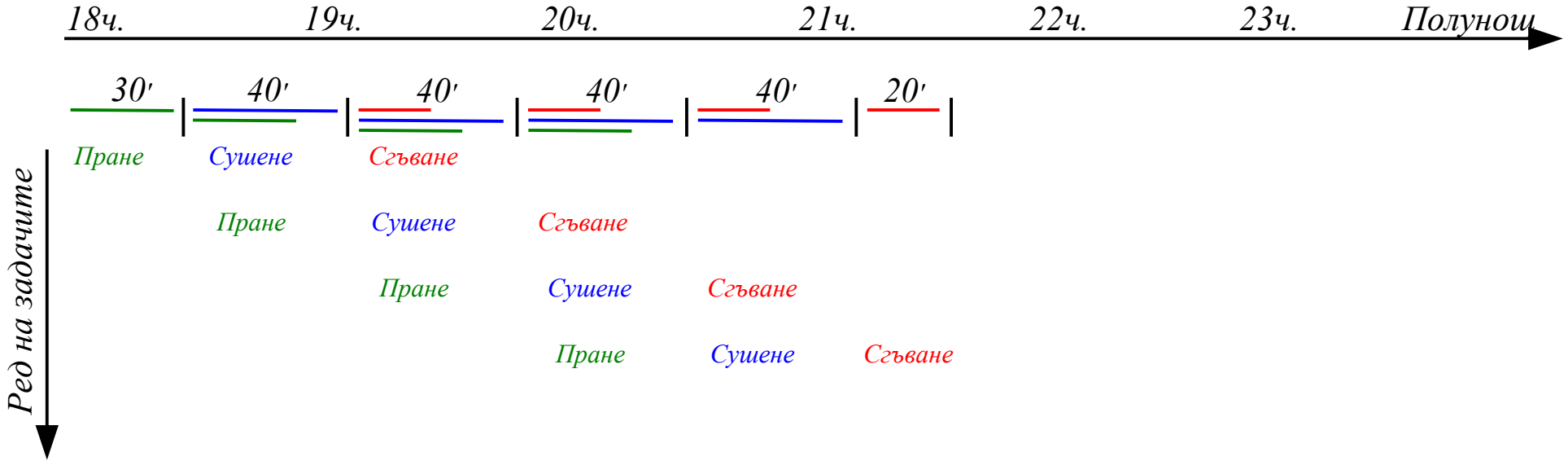
Пример

Последователно пране



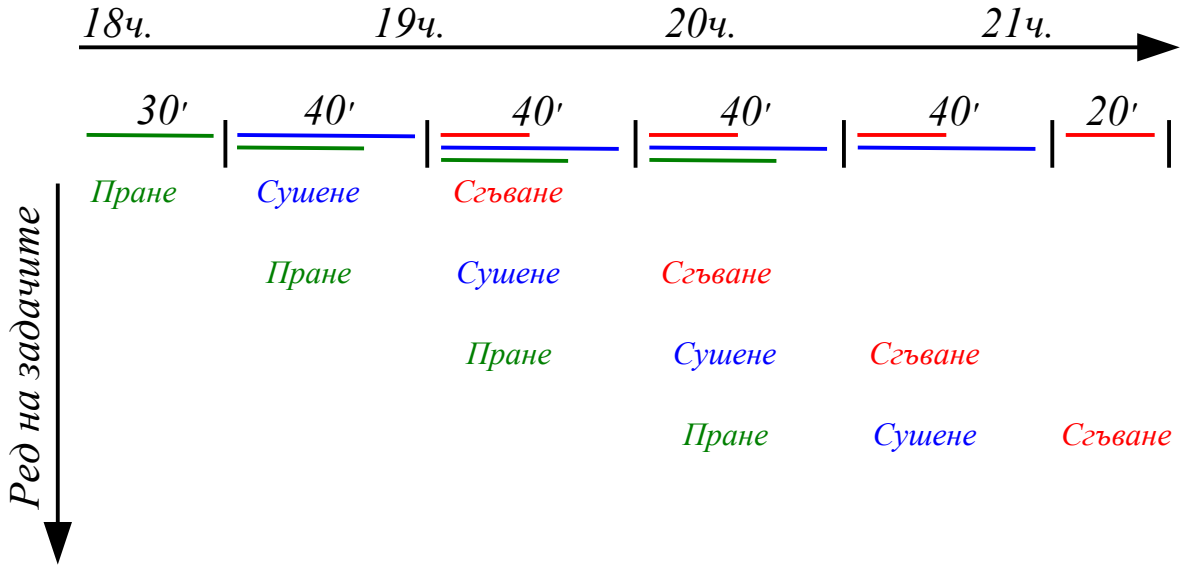
- Прането отнема 30 мин;
- Сушенето отнема 40 мин;
- Сгъването отнема 20 мин;
- Последователното пране отнема 6 часа за 4 купа дрехи;

Конвейерно пране



- Конвейерно означава задачата да се започва колкото се може по-скоро;
- Конвейерното пране отнема 3.5 часа!

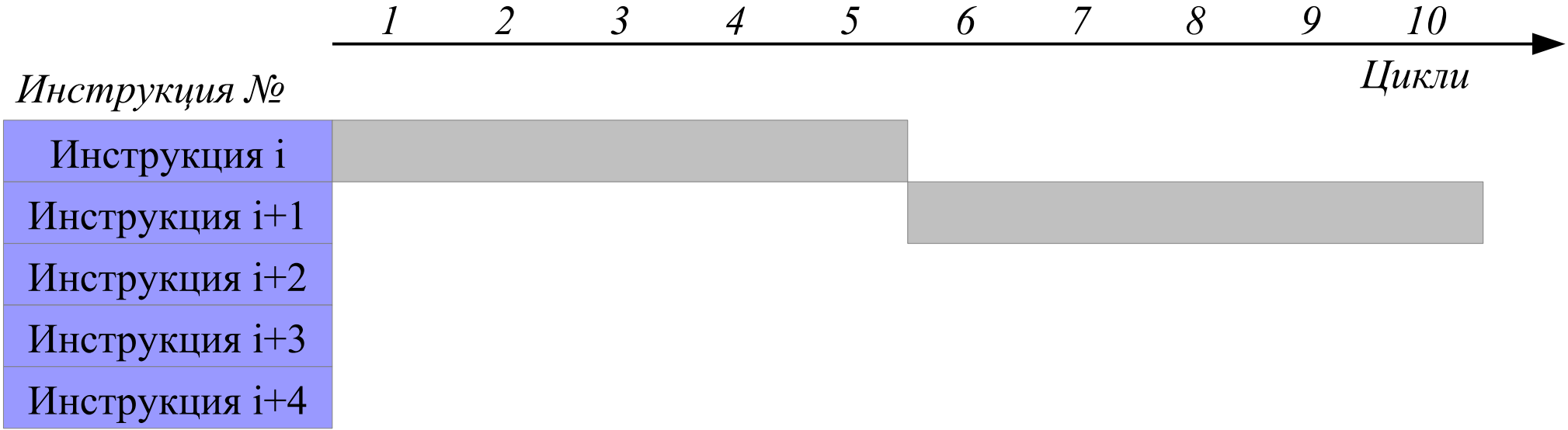
Изводи от конвейерното изпълнение



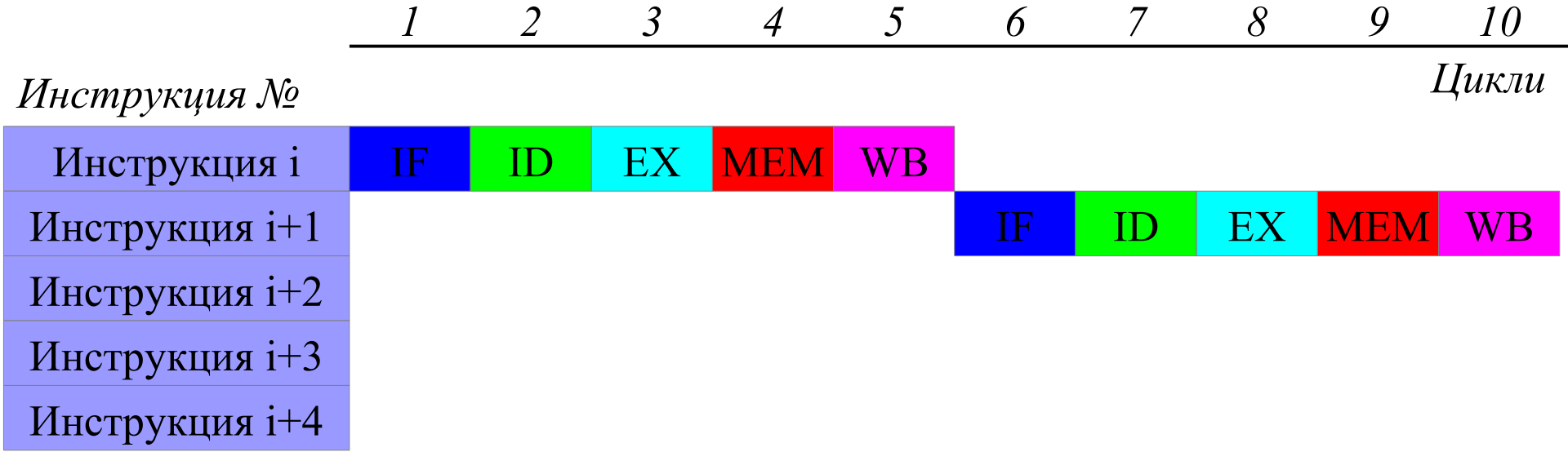
- Конвейерното изпълнение не намалява латентността на отделните задачи, а само пропускателната способност на цялото задание;
- Ефективността на конвейера е ограничена от най-бавната операция;

- Множествено задачи се изпълняват едновременно когато използват различни ресурси;
- Възможното ускорение е броя стъпки в конвейера;
- Небалансираните дължини на стъпките намаляват ускорението;
- Времето за напълване на конвейера и времето на изпразване намаляват ускорението;
- Stall при наличие на зависимости;

Изпълнение на инструкции

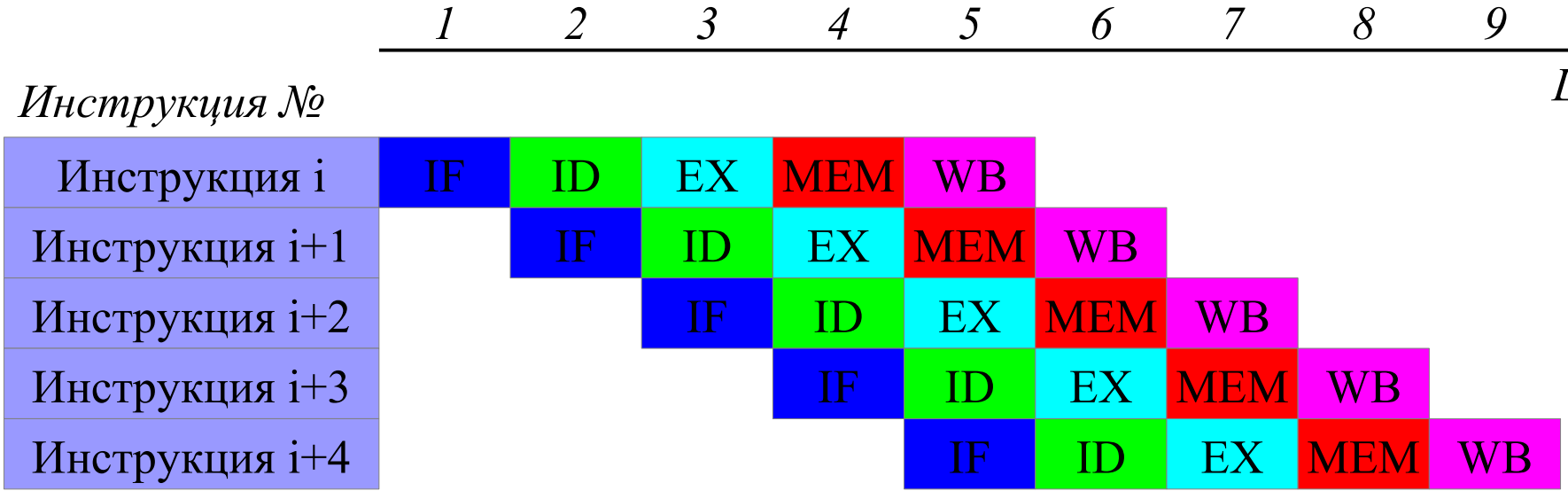


Изпълнение на инструкции



- IF – Instruction Fetch
- ID – Instruction Decode
- EX – Execution
- MEM – Memory Access
- WB – Write Back

Изпълнение на инструкции

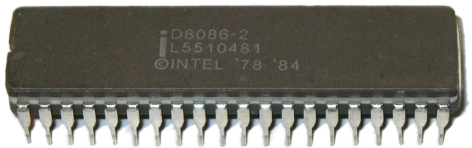


- IF – Instruction Fetch
- ID – Instruction Decode
- EX – Execution
- MEM – Memory Access
- WB – Write Back

Кратка История

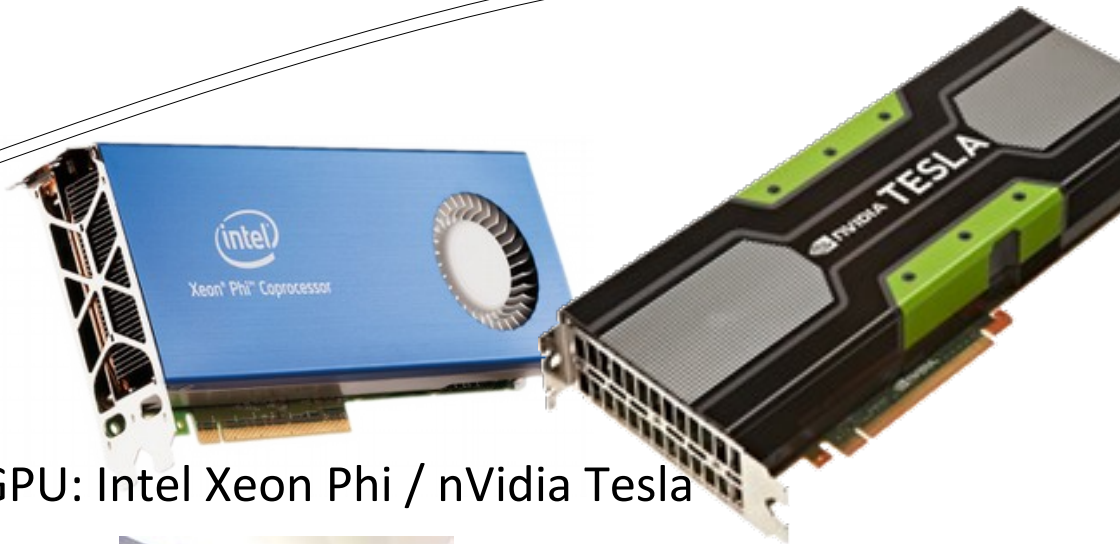
Защо?

Преди (70-те години на миналия век)

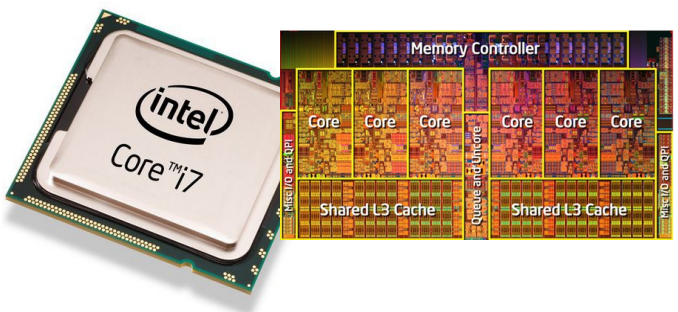


Intel 8086

Сега (2020 г.)



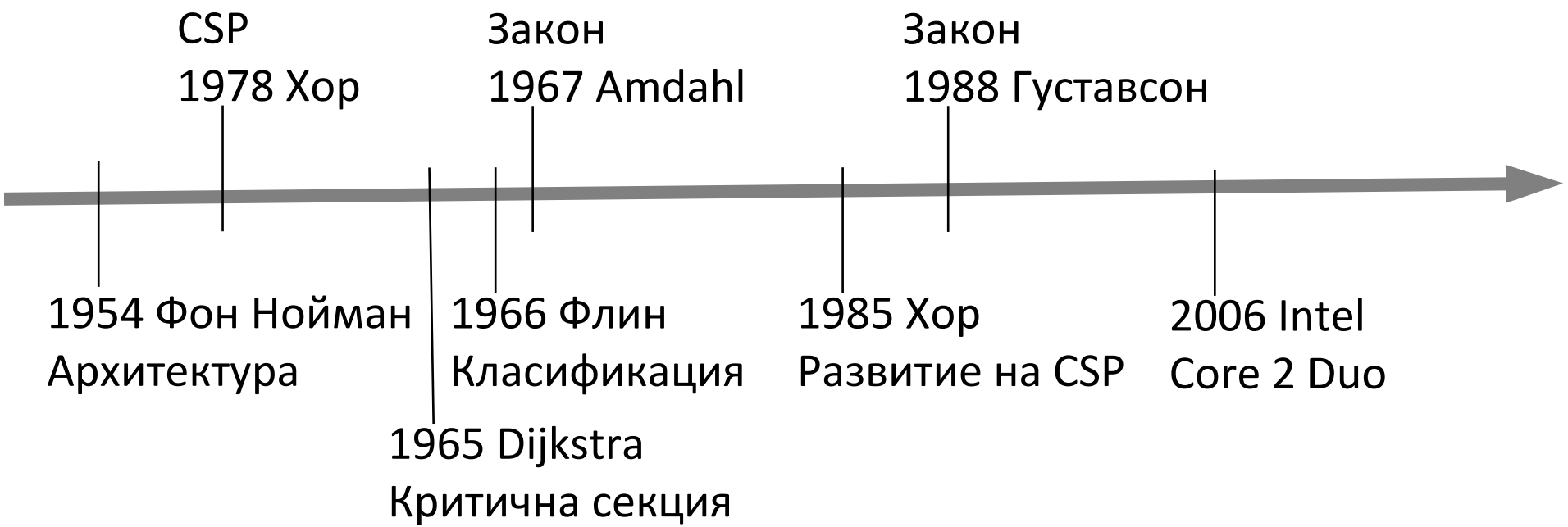
GPGPU: Intel Xeon Phi / nVidia Tesla



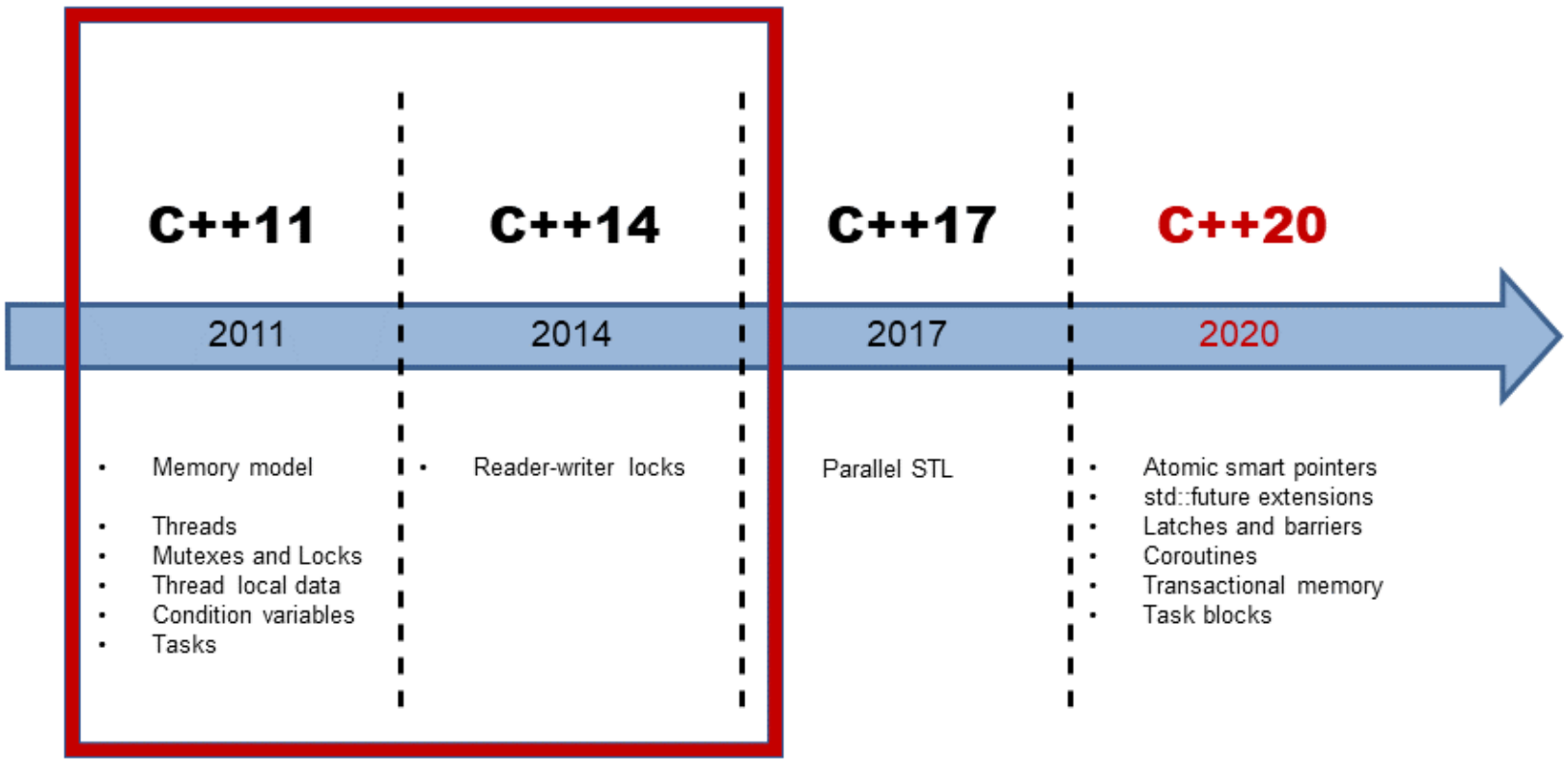
Intel i7 Процесор

Data Centers

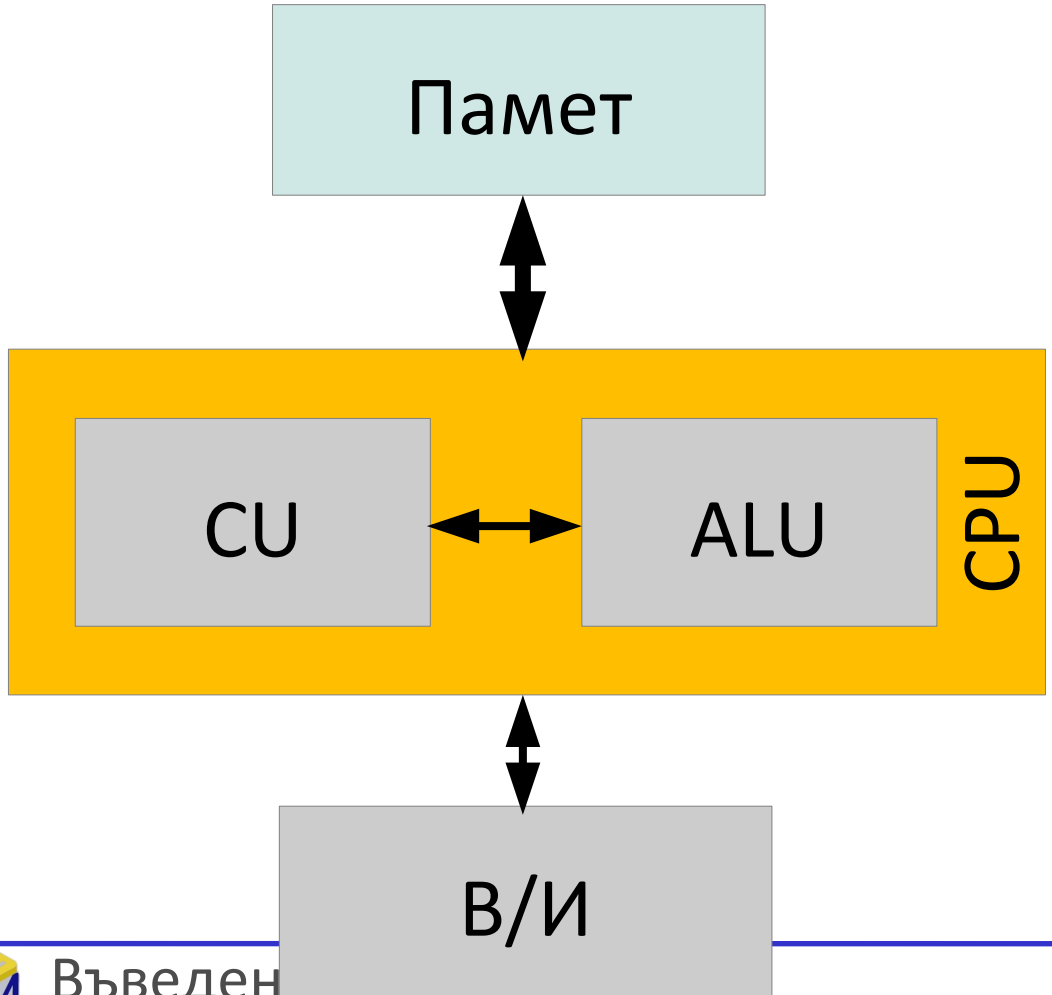
Времева линия



Времева линия (2)

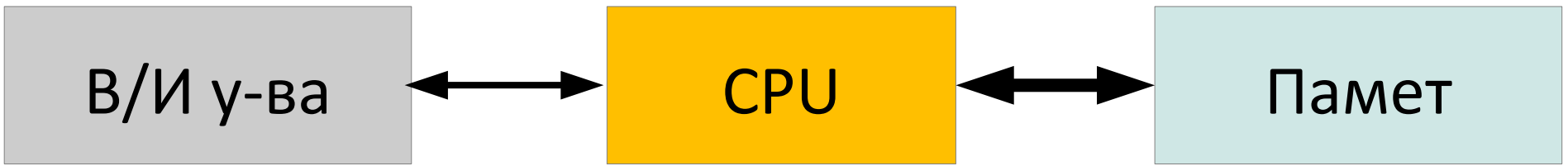


Фон Нойманова архитектура (1945)

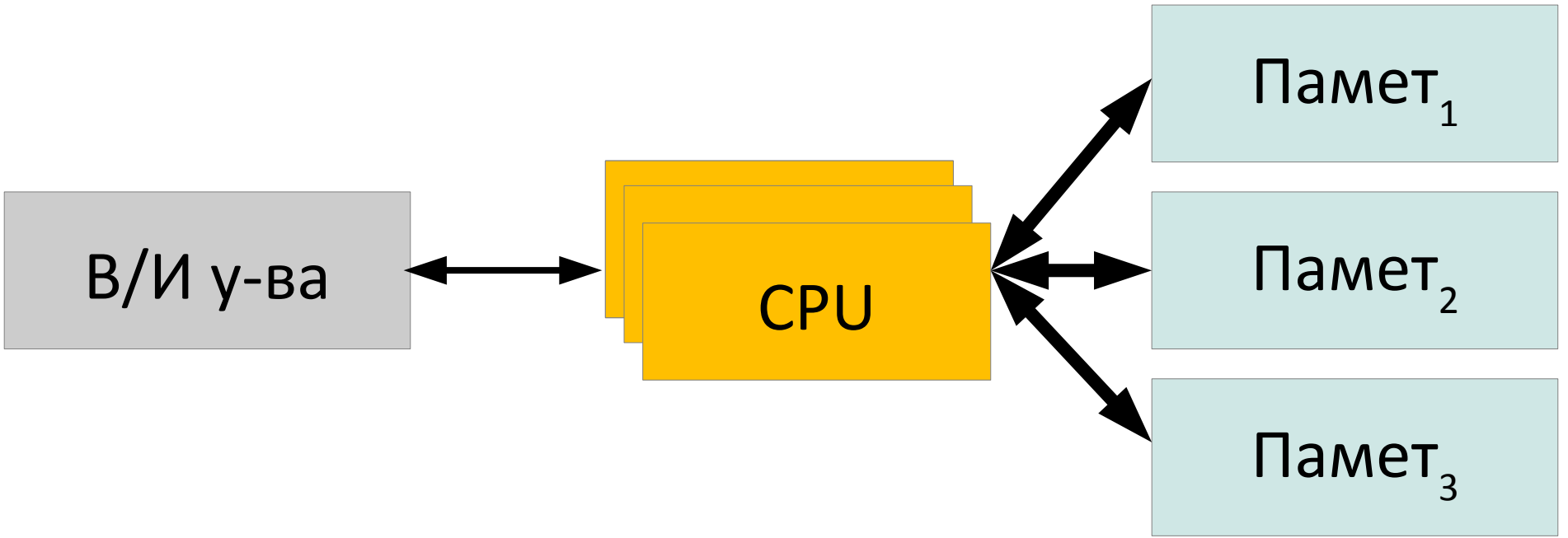


- Данните и Програмата се съхраняват в паметта;
- Контролното у-во (CU) извлича инстр. и данните от паметта, декодира ги и след това **последователно** координира изпълнението на операциите;
- Аритметичното и логическо у-во (ALU) извършва базовите аритметични операции;

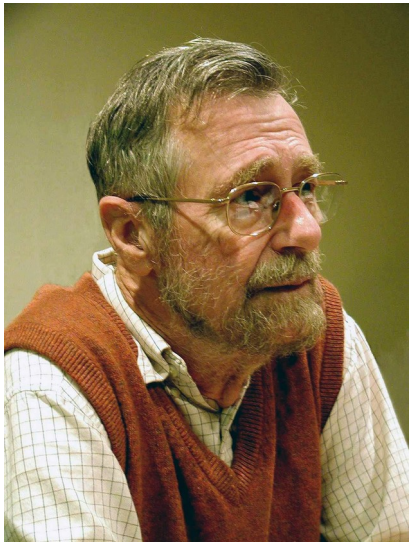
Класическа архитектура на компютър



Съвременен вариант на архитектурата



Дийкстра /Dijkstra/ (1965)



- За пръв път описва и дава име на критичните региони в паралелните алгоритми (Критична секция);
- Описва семафорите и проблема за „вечерящите философи” (1968);
- Въвежда понятието “защитени команди” (1975);

сър Тони Хор /Hoare/ (1978 и 1985)

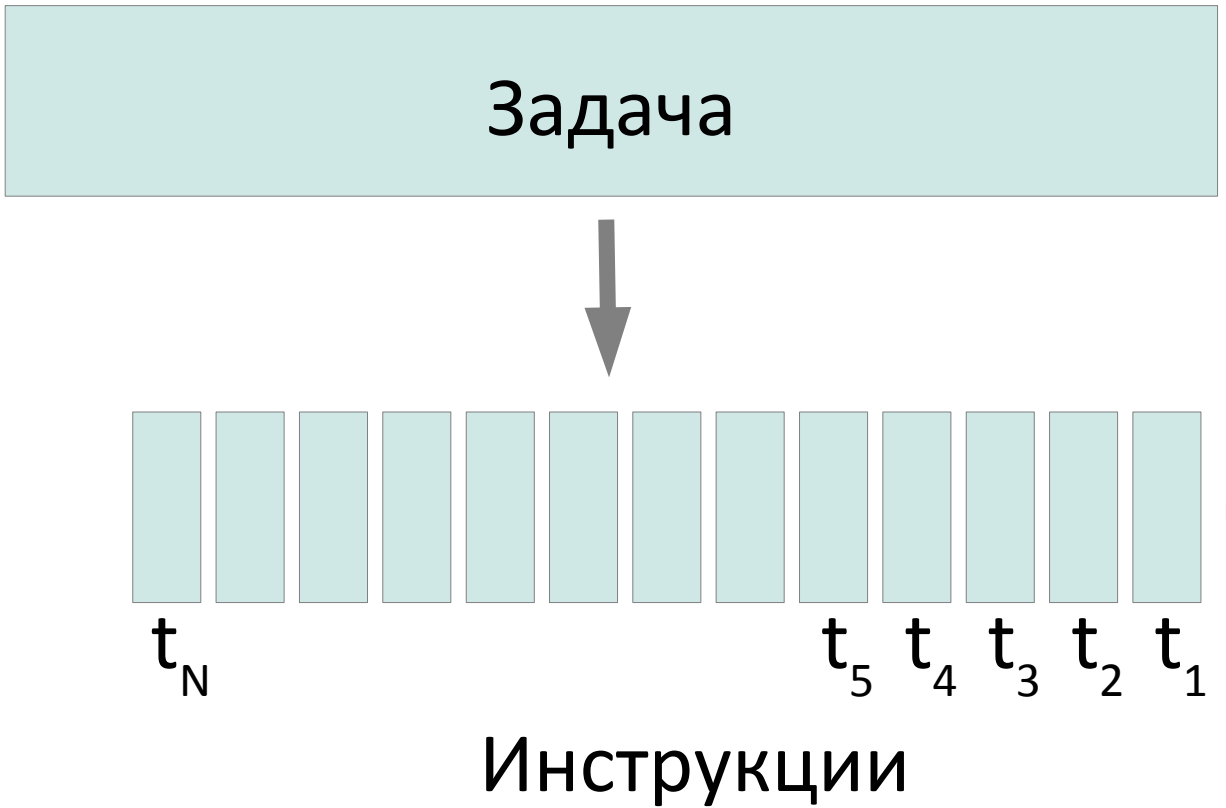
- Работата му по CSP (Communicating Sequential Processes) – описва шаблоните за взаимодействие на конкурентни системи;
- С тази си работа Хор става един от пионерите на теорията и практиката на паралелните системи;
- Това дава тласък на множество езици за паралелно програмиране като OCCAM, Go и др., както и на инструменти за анализ на паралелни системи и др.;

Базови Понятия и Концепции

Необходимост от Паралелни Архитектури (ПА) и Паралелно Програмиране (ПП)

- Физически ограничения пред скалируемостта;
  скорост, пространство, памет, енергия, ...
- Решаване на по-големи проблеми;
- Решаване на проблеми по-бързо;
- Решаване на проблеми, които „не пасват“ на едно CPU;

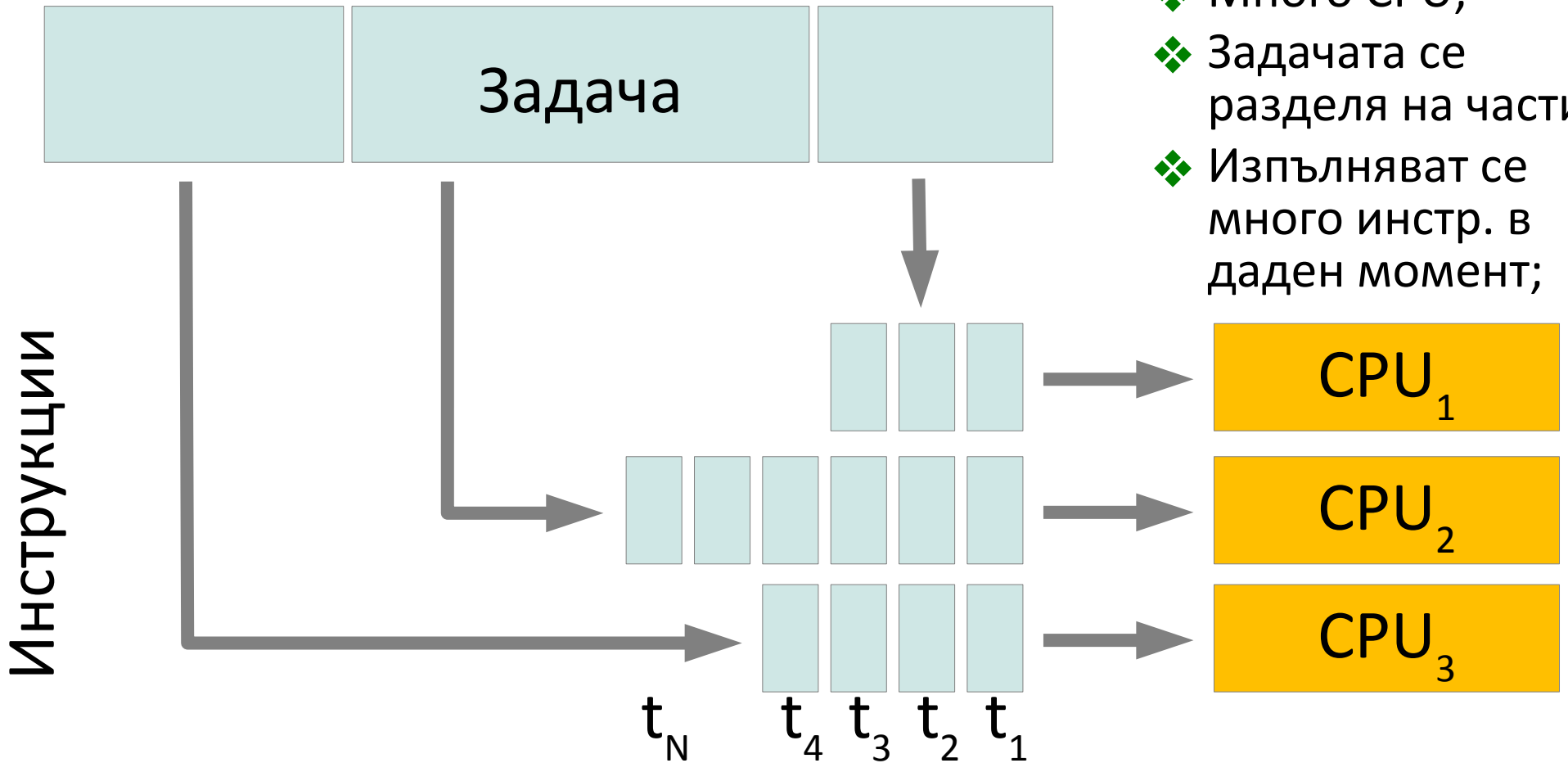
Не паралелно (серийно/последователно) изп.



- 1 CPU;
- 1 Инструкция в даден момент;
- Инстр. се изпълняват една след друга;

Паралелно изпълнение

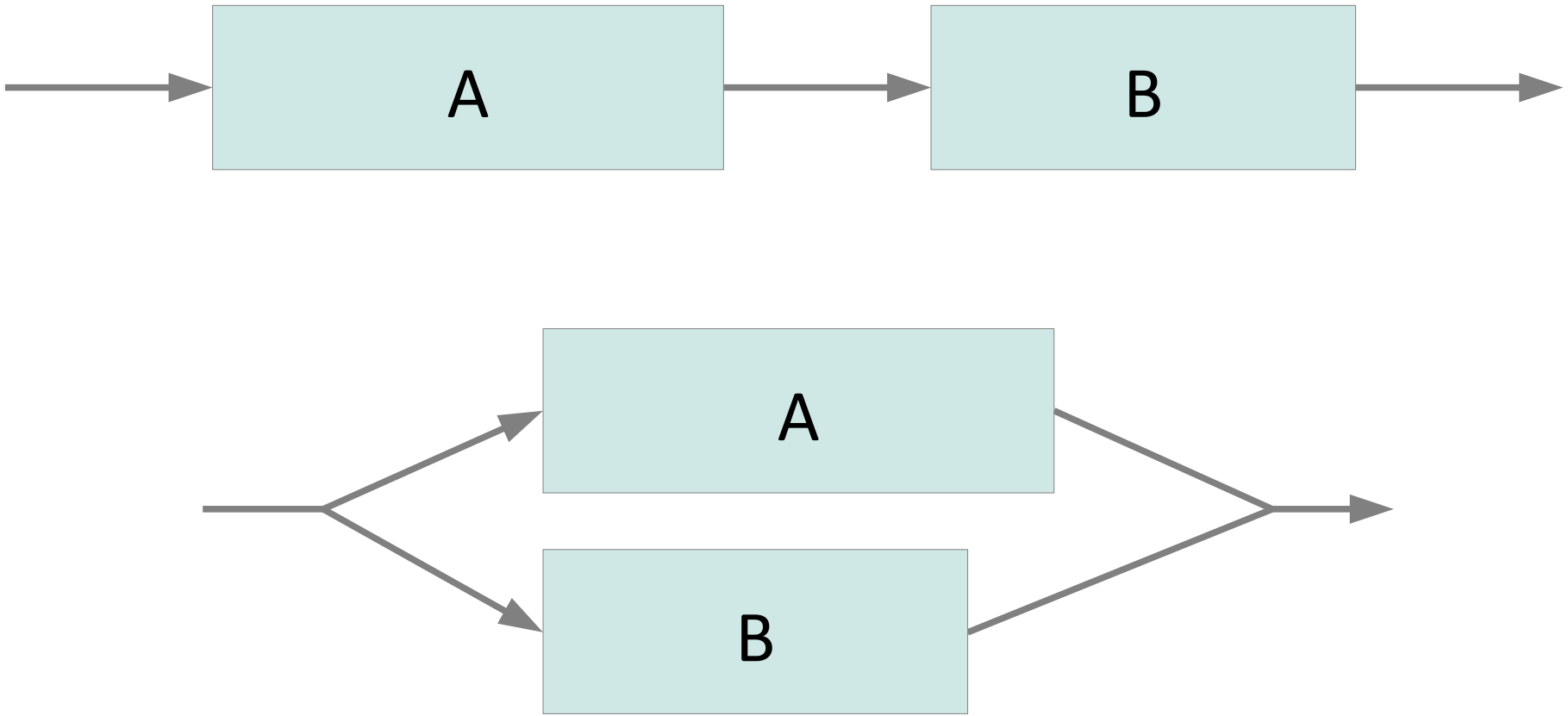
- Много CPU;
- Задачата се разделя на части;
- Изпълняват се много инстр. в даден момент;



Какво е паралелизация или разпаралеляване?

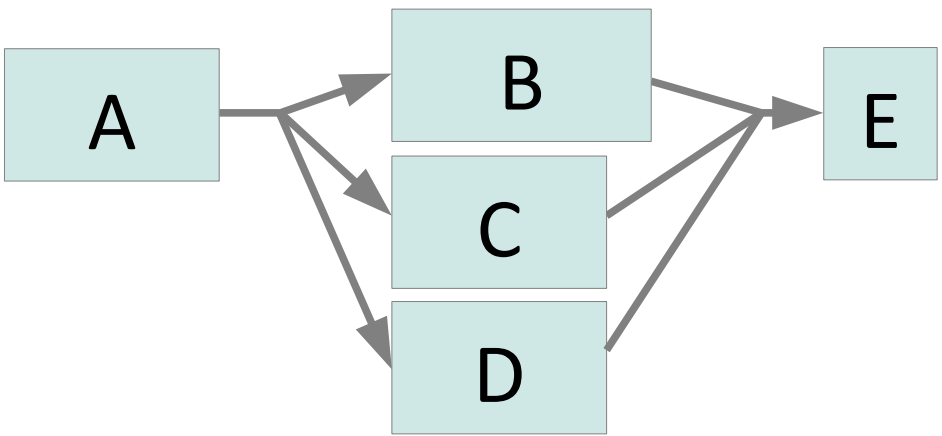
- Паралелизацията (parallelization) е процес на преобразуване на програма изпълняваща стъпките на алгоритмите си последователно, в програма изпълняваща ги (там където е възможно) паралелно/едновременно;
- Паралелното изпълнение може да е с използването на SIMD инструкции (векторизация), много нишки, много процеси, много компютри и т.н.;
- Обикновено се използва повече от едно ядро/процесор/изпълнител;

Какво е паралелизация?

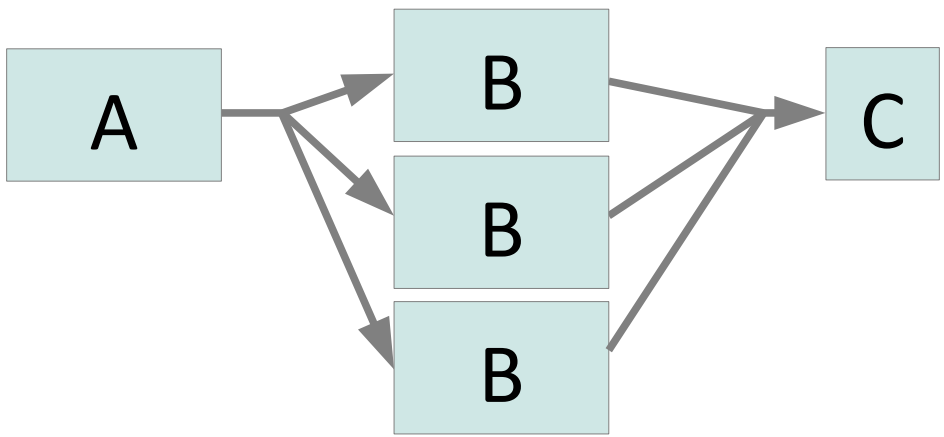


Функционален и Даннов Паралелизъм

- Функционален паралелизъм (Functional parallelism) – Всеки процесор работи върху част от проблема (задачата);
- Даннов паралелизъм (Data parallelism) – Всеки процесор извършва една и съща работа върху част от данните при решаването на проблема;

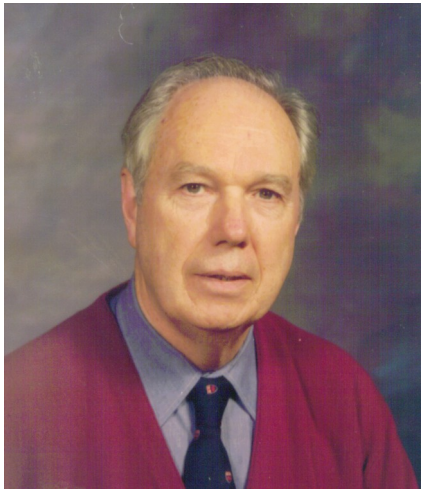


Функционален



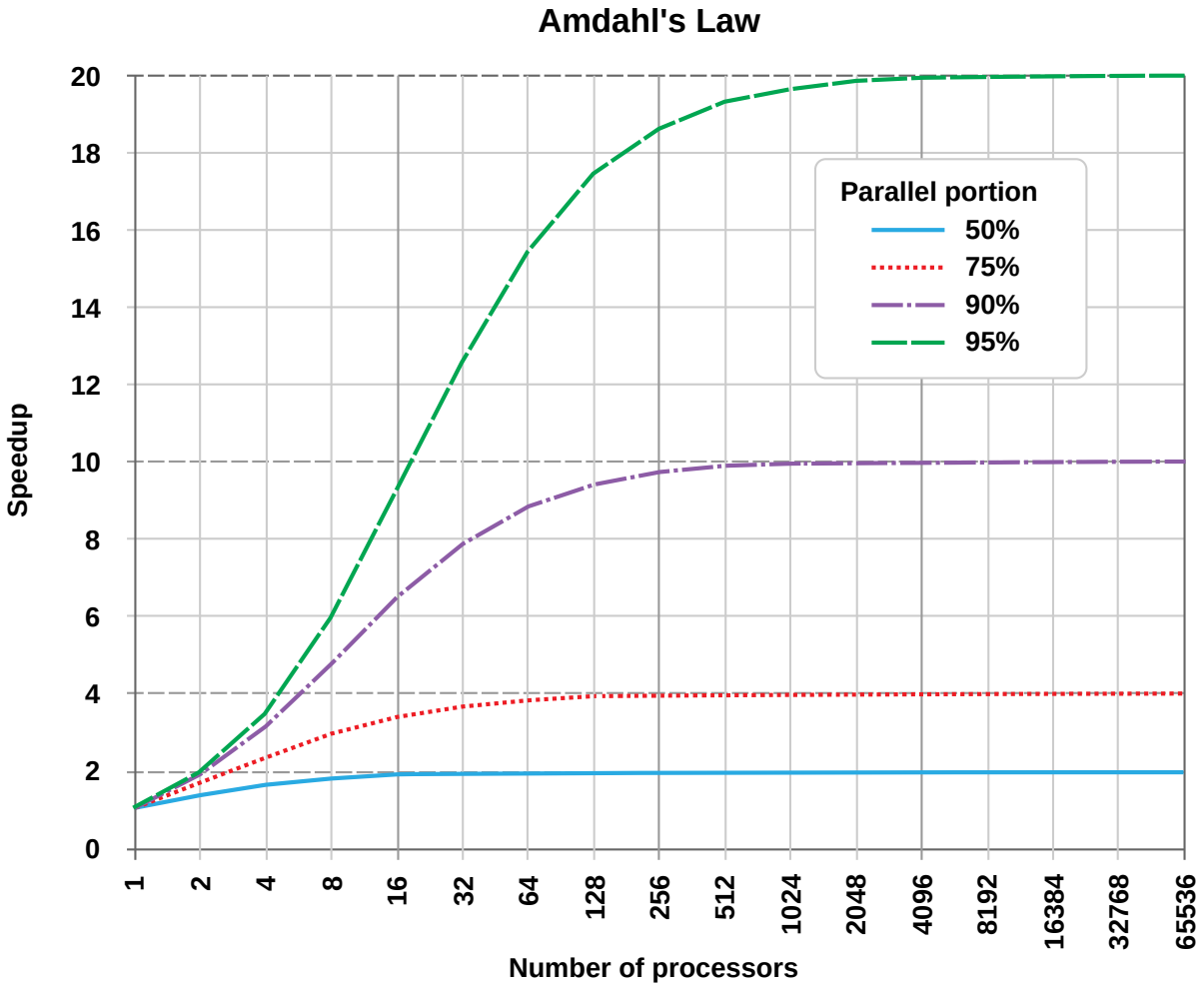
Даннов

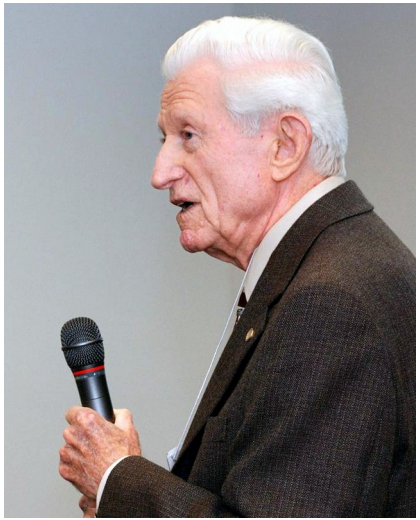
Класификация на Флин /Flynn/ (1966)



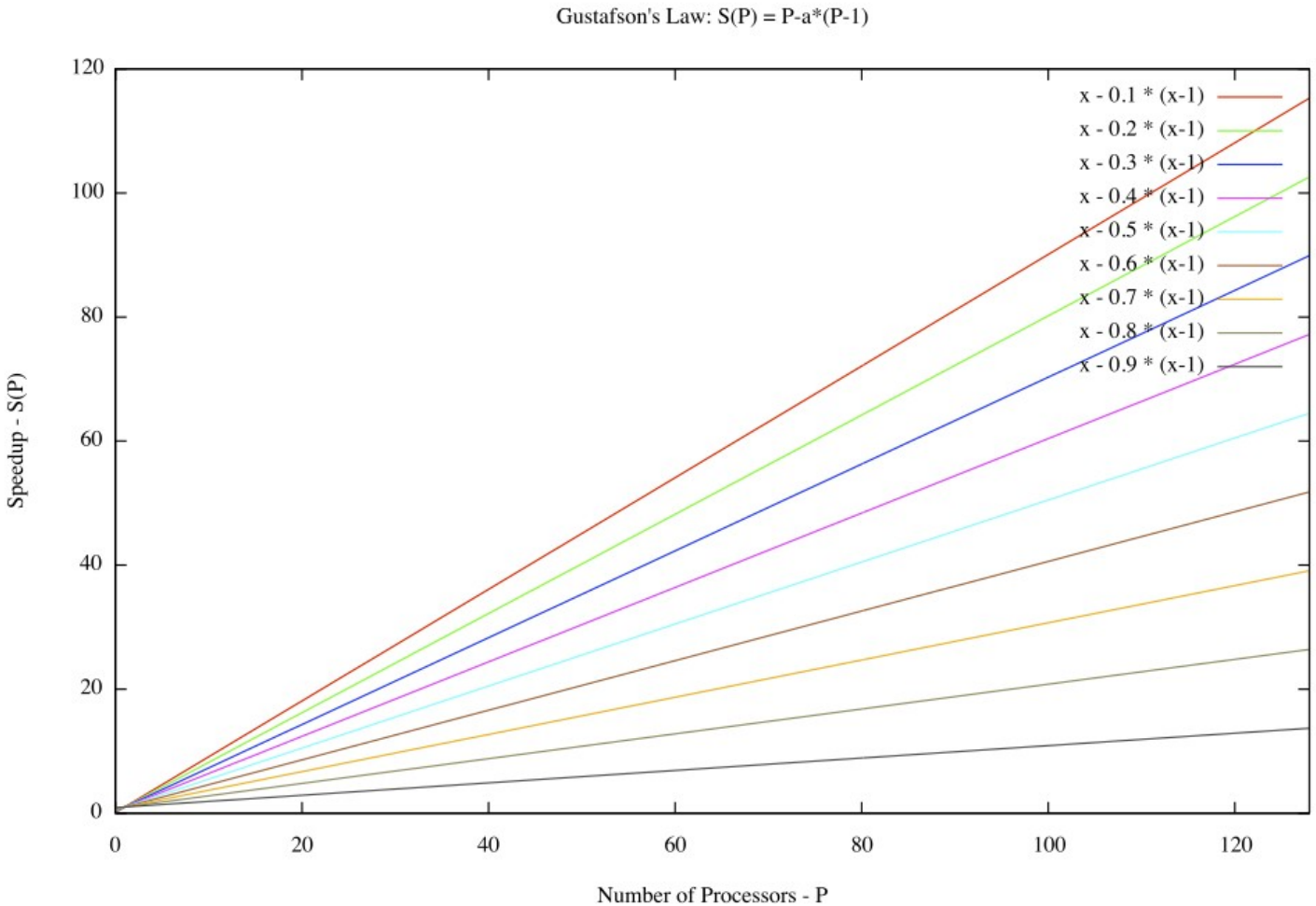
SISD Старите Mainframe, класическите PCи др.	**SIMD** Повечето съвременни PC, GPU и др.
MISD Използва за системи защитени от повреди, C.mmp computer и др.	**MIMD** Много-ядрените суперскаларни проц., разпределените системи и др.

Закон на Амдала /Amdahl/ (1967)





Закон на Густвсон /Gustafson/ (1988)



Паралелни архитектури

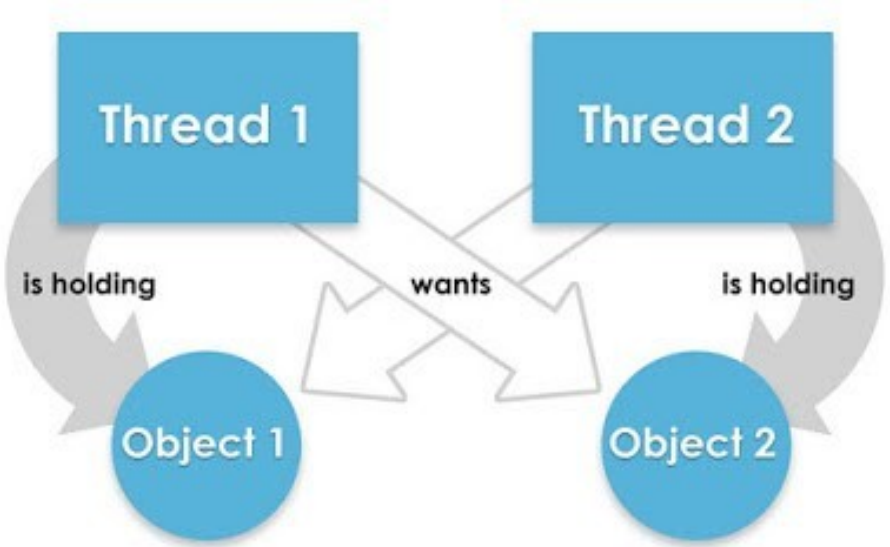
- Споделена памет (Shared memory)
  Много процесори могат да правят достъп до обща памет.
  - Еднороден достъп до паметта (Uniform memory access – UMA)
    Идентични процесори имат еднакъв достъп и еднакво време на достъп до паметта.
  - Не-еднороден достъп до паметта (Non-uniform memory access – NUMA)
    Не всички процесори имат еднакъв достъп и еднакво време на достъп до паметта.
  - Cache only memory architecture (COMA)
    Цялата памет на възлите се използва само като кеш.
- Разпределена памет (Distributed memory)
  Изисква комуникация по мрежата за достъп до между процесорната памет.
- Хибридна Разпределено-Споделена памет (Hybrid Distributed-Shared Memory)

Модели за Паралелно Програмиране

- Споделена памет (Shared memory)
  Задачите използват общо адресно пространство (обща памет), в която могат да пишат и четат асинхронно. Използват се различни механизми за защита и контрол на достъпа до общата памет
- Нишки (Threads)
  Нишките са подпрограми в главната програма. Те комуникират помежду си през глобална памет. Примери за този модел са Posix Threads, OpenMP и др,
- Предаване на съобщения (Message Passing)
  Множество от задачи, използващи своя собствена локална памет за изчисленията. Обмен на данни става чрез изпращане и получаване на съобщения. Трансфера на данни обикновено е кооперативен т.е. изпращащата страна трябва да има получател. Пример: MPI
- Данново паралелен (Data Parallel)
  Множество от задачи, работещи колективно върху обща структура от данни. Всяка задача извършва едно и също действие със своята част от данните
- Хибриден (Hybrid)

Проблеми при ПП и Координация в Паралелни Алгоритми (ПАлг.)

- Мъртва хватка (Deadlock);
- Жива хватка (Livelock);
- Трудна скалируемост;
- Глад за ресурси;
- Съперничество;
- др.



Пример – критична скеция

Може да възникне проблем при паралелно изпълнение

```
void add(hashtable table, value v) {
  int h = hash(v.key);
  v.next = table[h].next;
  table[h].next = &v;
}
```

Thread 1

```
h1 = hash(x.key);
x.next = table[h1].next;

table[h1].next = &x;
```

Thread 2

```
h2 = hash(y.key);

y.next = table[h2].next;

table[h2].next = &y;
```

Какъв е проблемът тук?

Пример – използване на критична секция

```
void add(hashtable table, value v) {
  int h = hash(v.key);
  mutex m;
  m.lock();
  v.next = table[h].next;
  table[h].next = &v;
  m.unlock();
}
```

Критична секция

Thread 1

```
h1 = hash(x.key);
m.lock();
x.next = table[h1].next;
table[h1].next = &x;
m.unlock();
```

Thread 2

```
h2 = hash(y.key);



m.lock();
y.next = table[h2].next;
table[h2].next = &y;
m.unlock();
```

Няма проблем при h1=h2, обаче...

Какъв е проблема при тази реализация на критичната секция?

Пример – по-добро използв. на критична секция

```
void add(hashtable table, value v) {
  int h = hash(v.key);
  table[h].mutex.lock();
  v.next = table[h].next;
  table[h].next = &v;
  table[h].mutex.unlock();
}
```

Thread 1

```
h1 = hash(x.key);
table[h1].mutex.lock();
x.next = table[h1].next;
table[h1].next = &x;
table[h1].mutex.unlock();
```

Thread 2

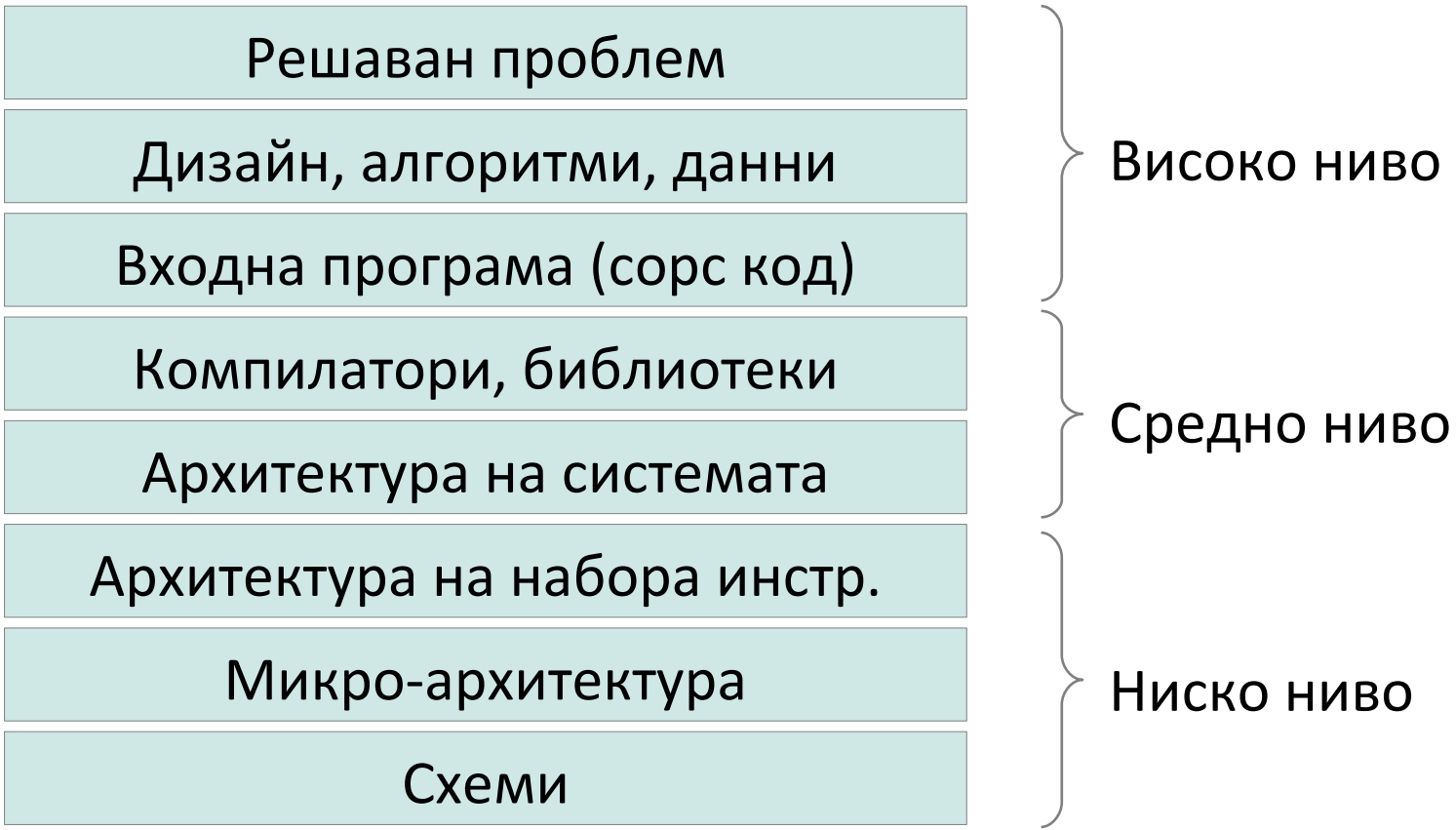
```
h2 = hash(y.key);
table[h2].mutex.lock();
y.next = table[h2].next;
table[h2].next = &y;
table[h2].mutex.unlock();
```

Така е по-добре, защото няма заключване при $h1 \neq h2$

Дизайн на Паралелни програми

- Анализ на проблема:
  - Може ли проблема да се разпаралели?
  - Има ли даннови зависимости?
  - Идентификация на „тесните“ места.
- Реализация:
  - Как да се разпределят данните?
  - Как да се разпределят инструкциите?
  - Каква ще е комуникацията?
  - Необходима ли е синхронизация (защита)?
    - Можем ли да използваме Fork-Join, Map-and-Reduce или други подобни известни подходи?
- Тестване – работоспособност, ефективност, скалируемост...;

Обща схема – нива на абстрактност



Проект

Курсов проект

Получавате неефективно работещ проект, който не използва техники от паралелното програмиране.

Вашата задача е:

- Паралелизация:
  - Преработвате проекта;
  - Прилагате техники от ПП;
  - Анализирате и подобрявате производителността;
- Оптимизация:
  - Възползвате се от неефективността на алгоритмите;
  - Възползвате се от ресурсите на наличната машина(и).
- Документация.

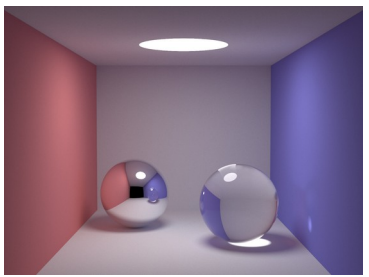
Курсов проект

Имате избор от две възможни теми*:

- Умножение на две много големи квадратни матрици;

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix} . \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1N} \\ b_{21} & b_{22} & \cdots & b_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & \cdots & b_{NN} \end{pmatrix} = ?$$

- Визуализация на 3D сцени с помощта на Ray-tracing;



Няма единствен „верен отговор“:

- Имате много свобода (и малко напътствия);
- Обаче има лесен начин да се види кой е най-добрият;

* Виж примерните проекти в сайта с помощни материали

Политика за работа (можете да)

- Използвате произволен език за програмиране;
- Използвате произволна среда за разработка;
- Използвате произволна операционната система;
- Използвате произволни методи и библиотеки за паралелно програмиране и оптимизация;
- Използвате примерните проекти (виж помощните материали), като основа за създаване на собствен проект или като пример за не паралелна реализация на съответната задача;
- Всеки работи самостоятелно;
- Допустимо е да обсъждате проблема, изискванията и материалите с всеки;

Политика за работа (не можете да)

- Не може да използвате проекти (или части от тях) създадени от други лица за целите на този курс, включително създадени в предишни учебни години;
- Не може да използвате проекти (или части от тях) създадени от други лица за сходни цели;
- Не може да използвате код, който не сте реализирали собственоръчно (с изключение на код от примерните проекти);

Предаване на проекта

Всеки трябва да работи по проекта си по време на семестъра и да предаде проекта си
(сорс код + документация описваща направените промени)
в хранилището си в SVN подпапка **/trunk/PPProject**
поне **два дни** преди изпита.

Успех!