IVS

Prezentace přednášek

Ústav počítačové grafiky a multimédií

Urychlování výpočtů,

možnosti paralelizace

Motto

„If Edison had a needle to find in a haystack, he would proceed at once with the diligence of the bee to examine straw after straw until he found the object of his search.
I was a sorry witness of such doings, knowing that a little theory and calculation would have saved him ninety per cent of his labor.“

- Nikola Tesla

Obsah:

- Motivace
- Základní strategie urychlování výpočtů
- Paralelizace, Amdahlův a Gustafsonův zákon
- Paralelizmus
- Datové závislosti
- Synchronizace, Hazardy
- Paralelní architektury
- Paralelní programování
- Příklady
- Literatura

Motivace

- Některé výpočty trvají „příliš dlouho“
  - Jednoduše nechceme čekat, chceme výsledek „hned“

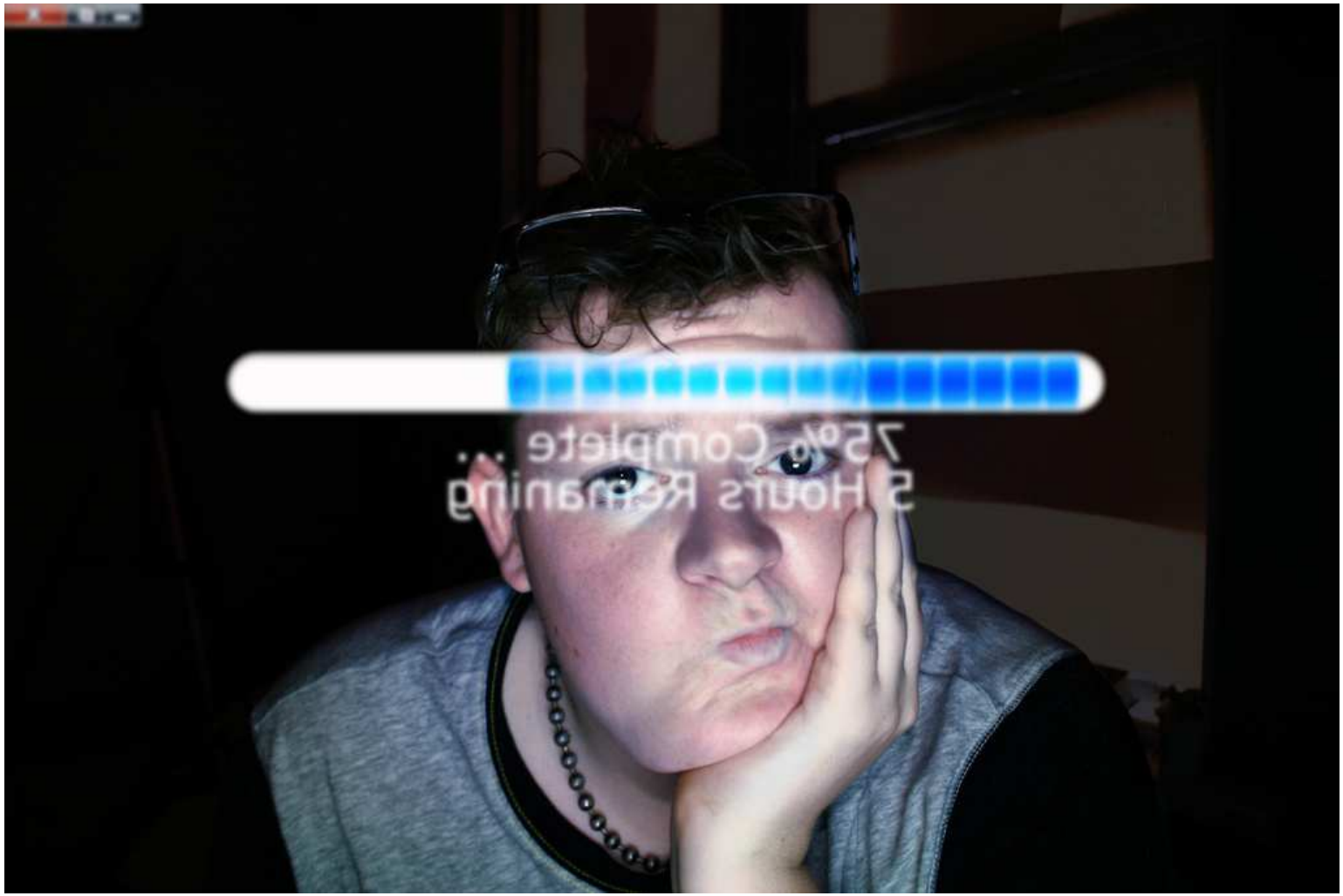


image by Sam Davis

Motivace

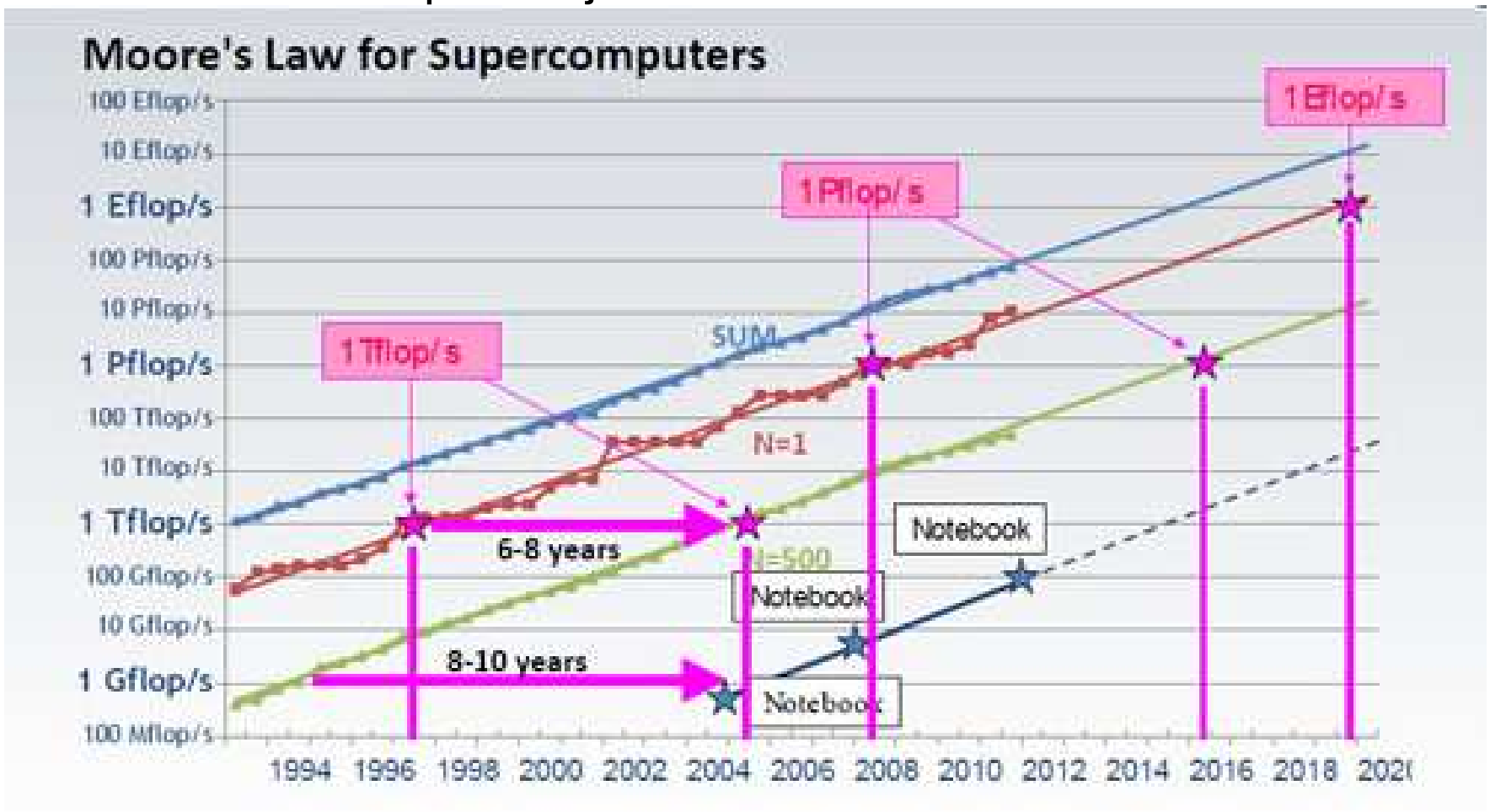
- Některé výpočty jsou součástí delšího ***řetězce*** zpracování
  - Musíme stíhat zpracovávat data jak přicházejí
  - Např. předpověď počasí, LHC, SETI

Motivace

- Výpočty běžící v ***reálném čase***
  - Výpočet je omezený pevně daným časovým intervalem
  - Např. dekódování videa – musí stíhat min 25 snímků za vteřinu

Motivace

- Moorův zákon
  - počet tranzistorů se každé dva roky zdvojnásobí
  - trochu se nám zpomaluje



Základní strategie urychlování výpočtů

- Použití optimálních algoritmů
  - Složitost algoritmů jako funkce velikosti vstupních dat
  - Pro základní algoritmy známe jejich optimální složitost
  - Značíme ***O(n)***

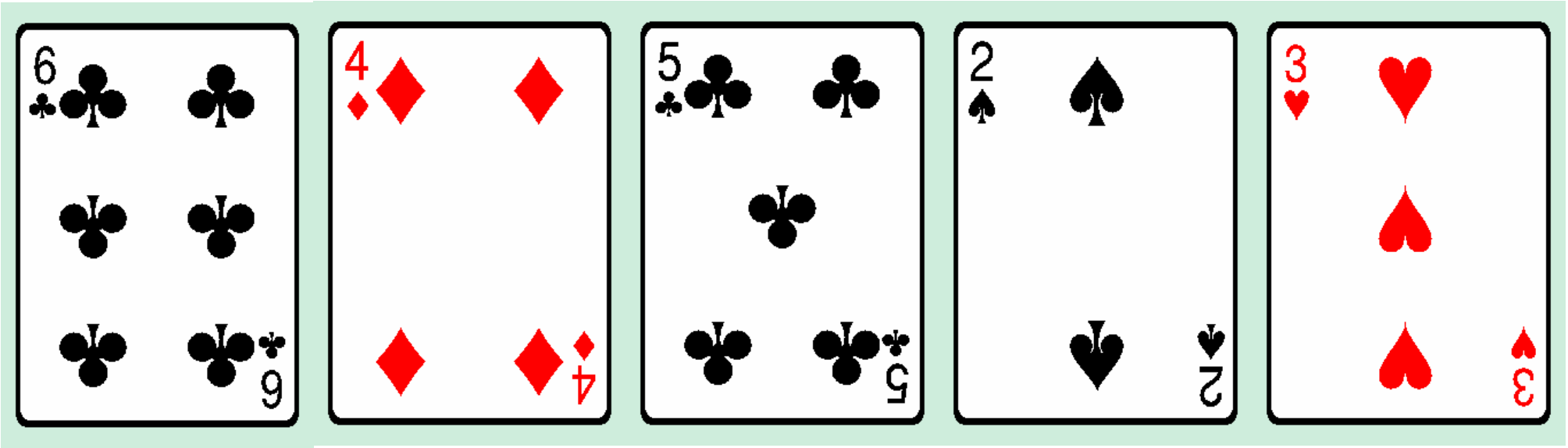
Sorting Algorithm Explained with a Hungarian Folk Dance

Základní strategie urychlování výpočtů

- ***BozoSort*** má (worst case) složitost ***O(n·n!)*** ***faktoriální***

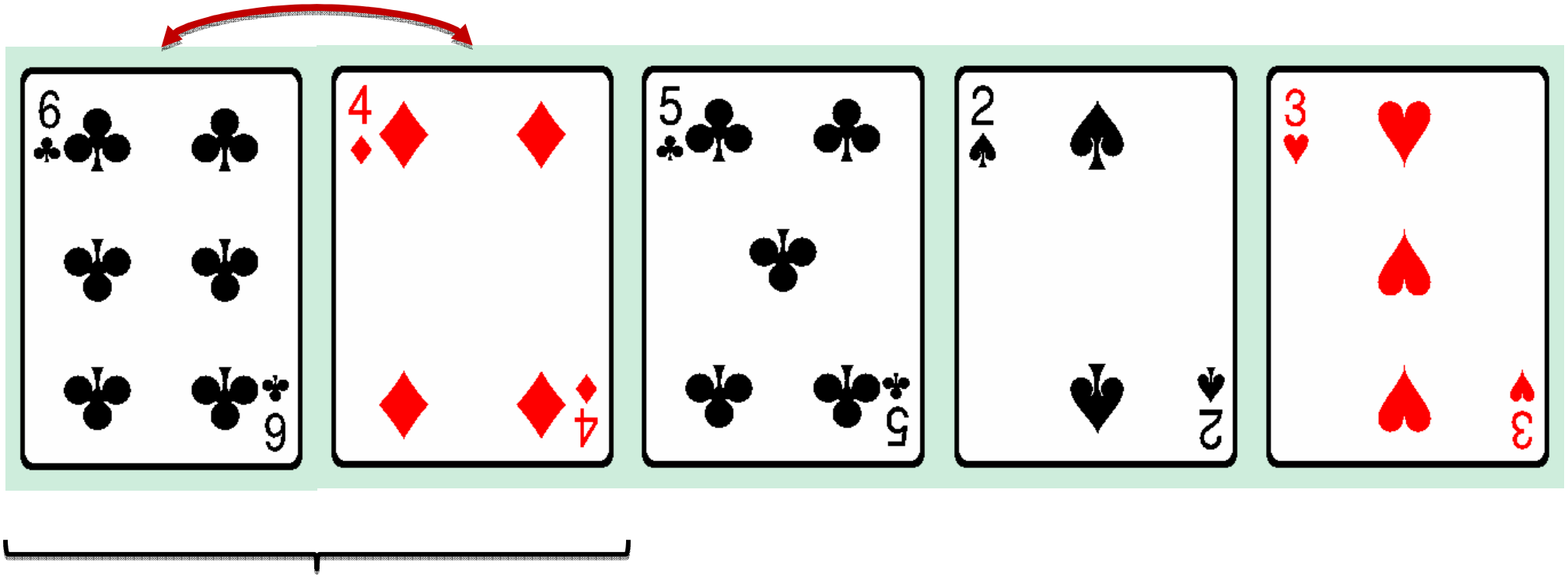
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



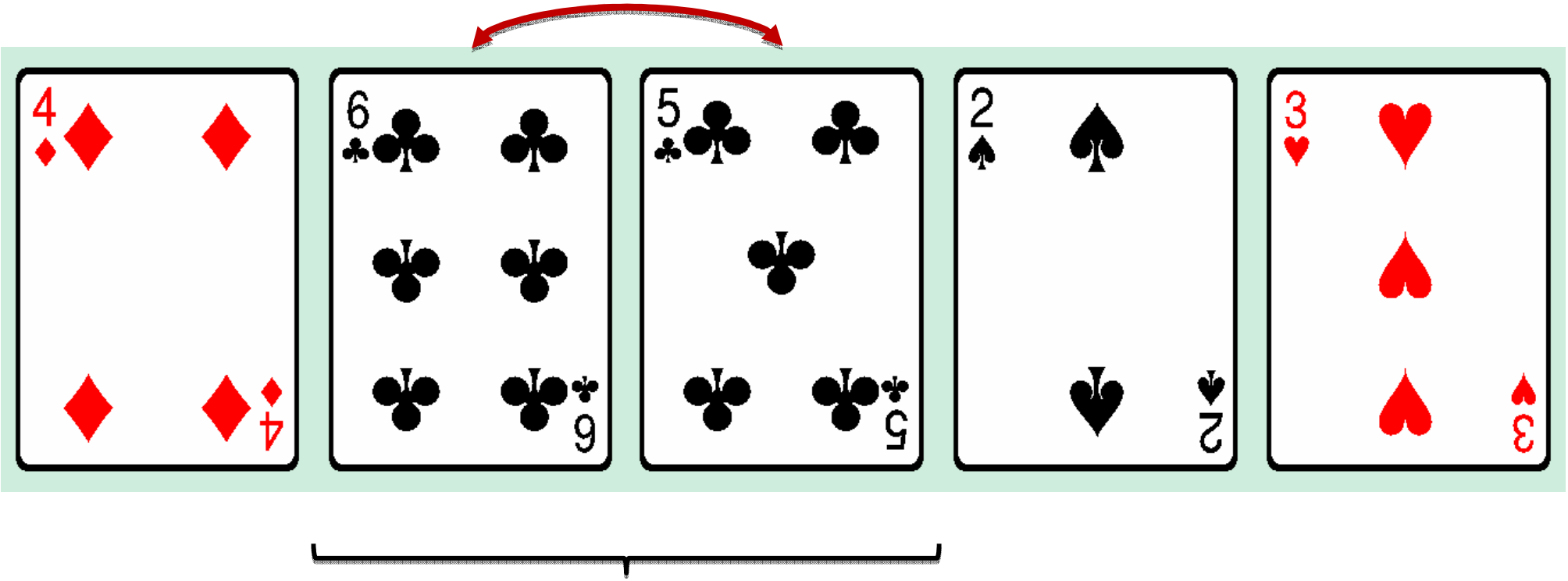
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



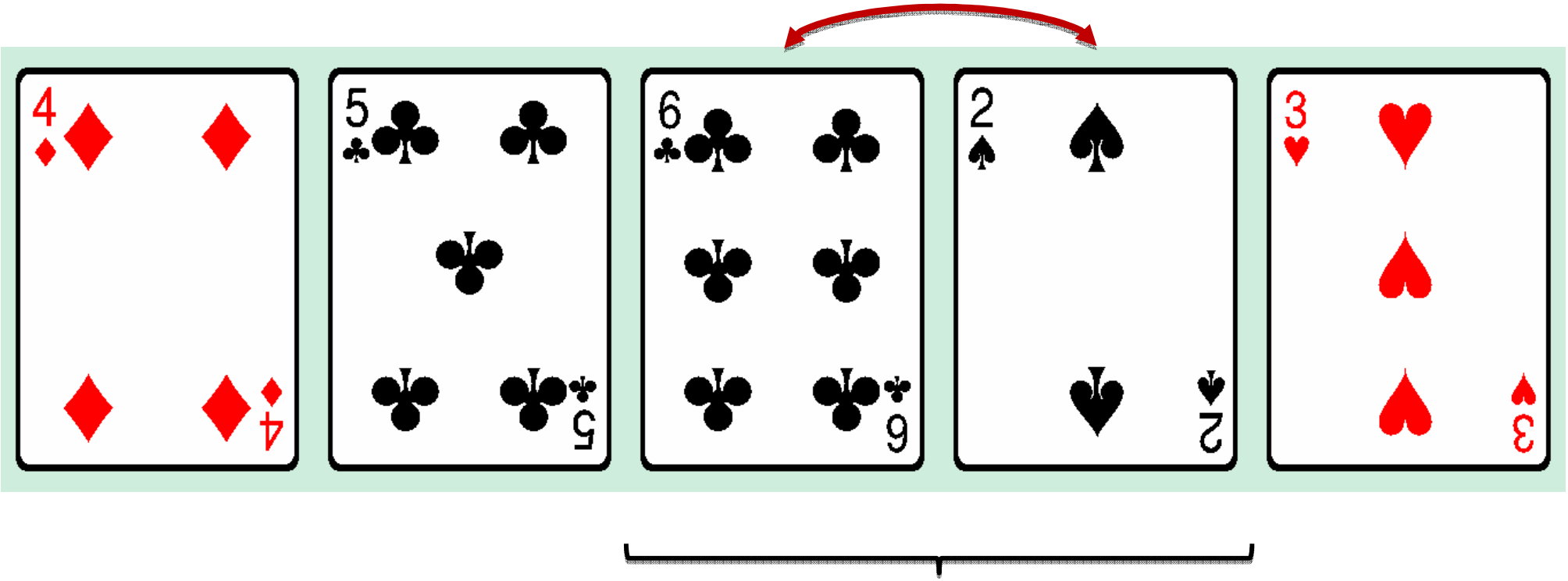
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



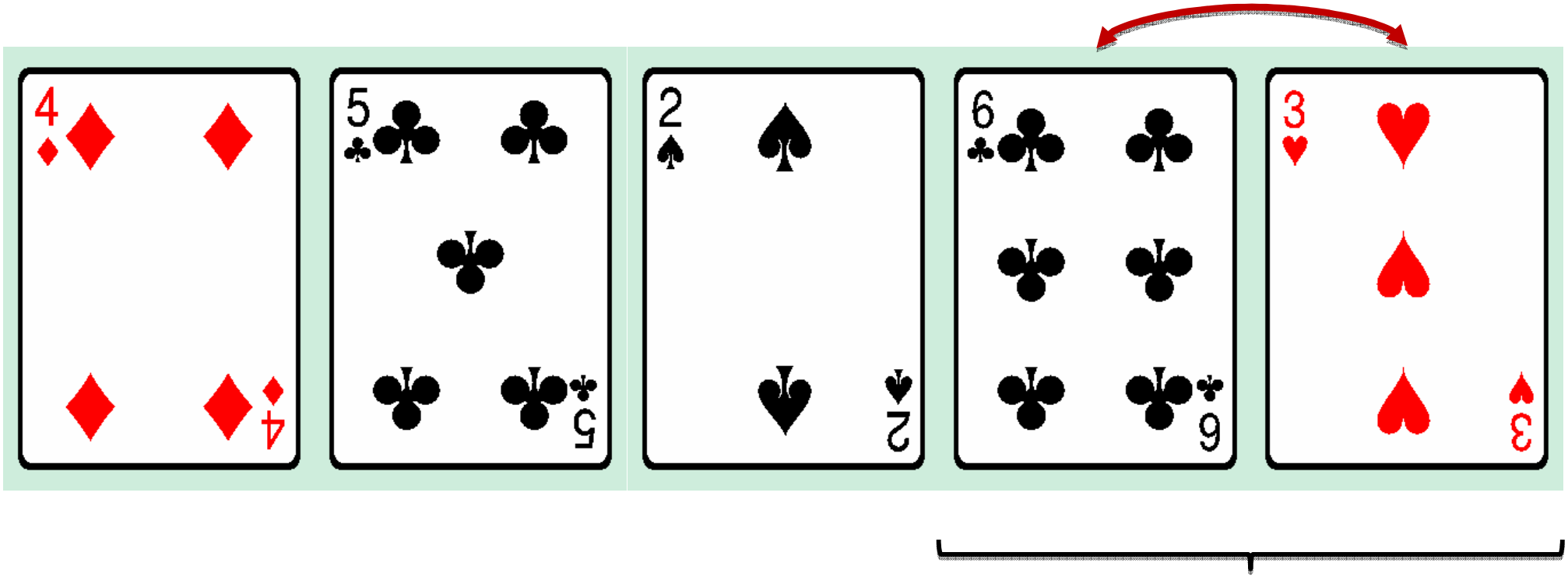
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



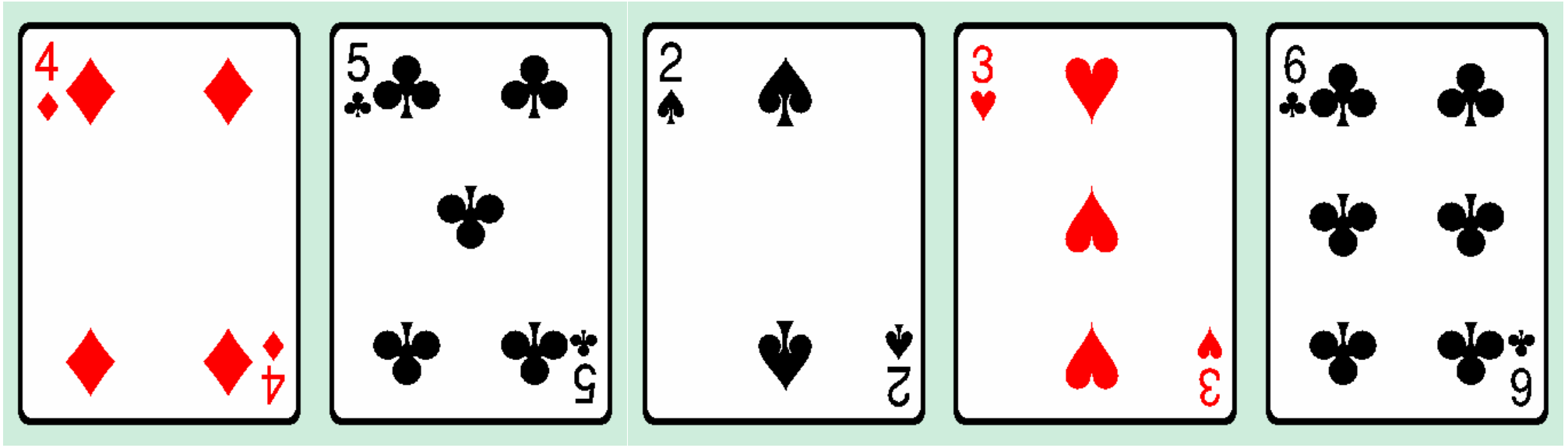
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



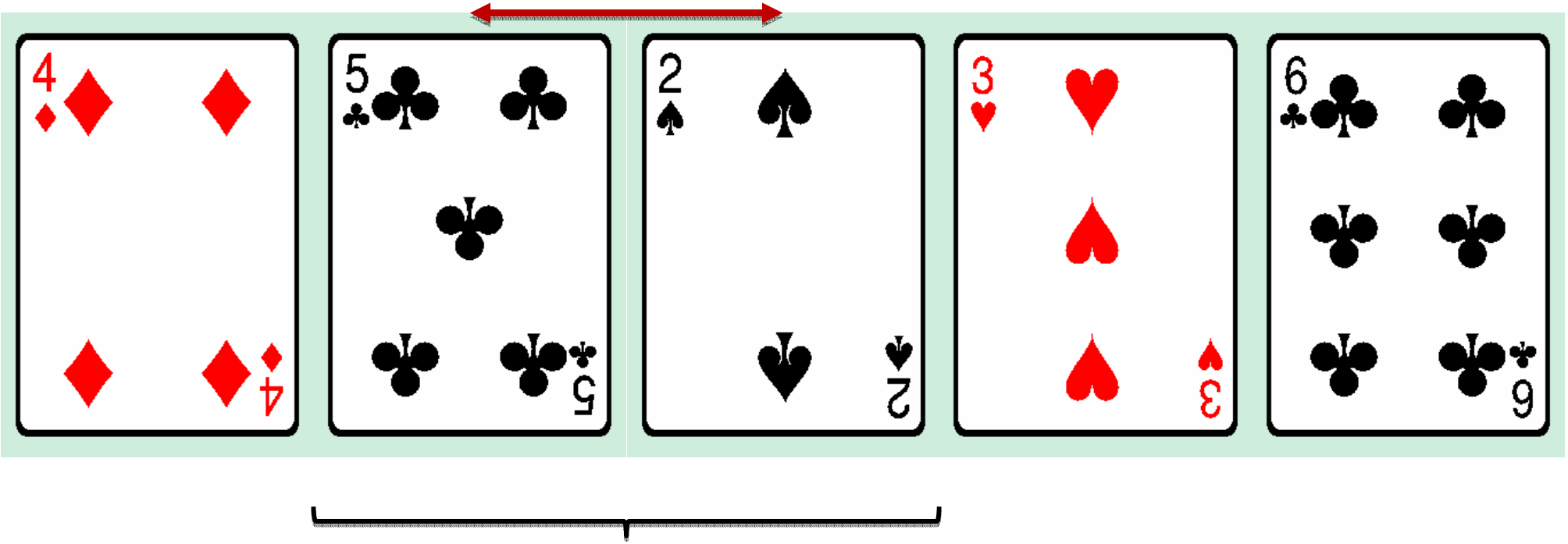
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



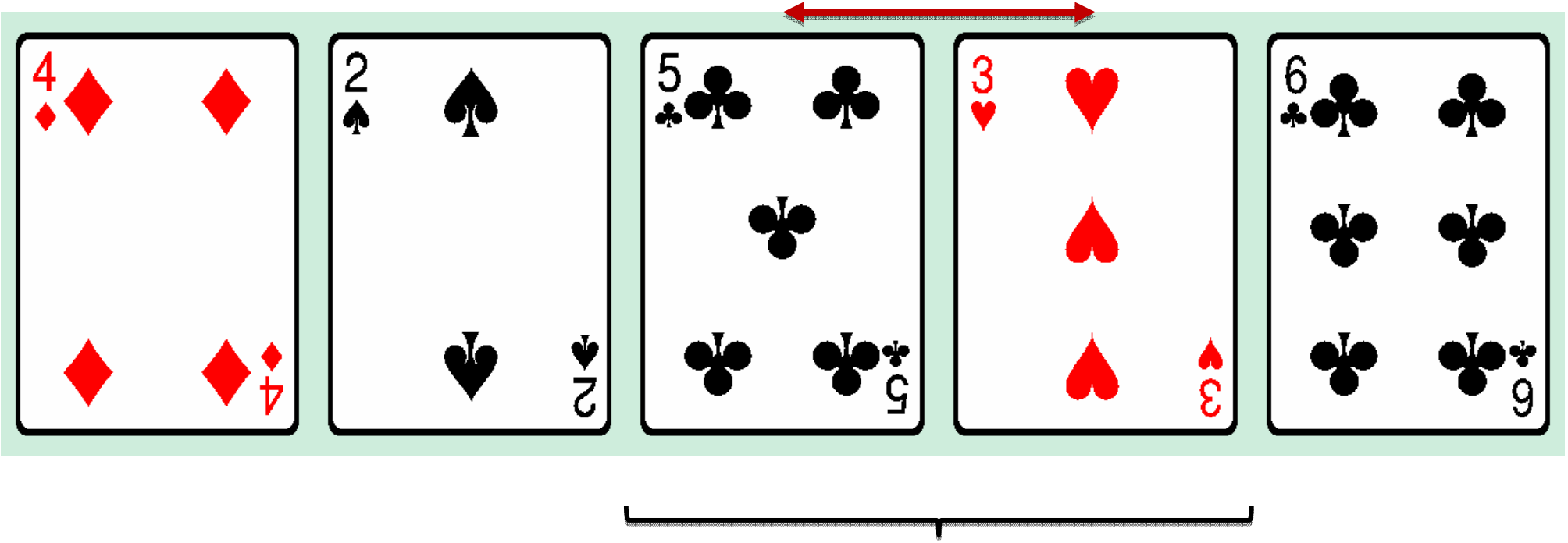
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



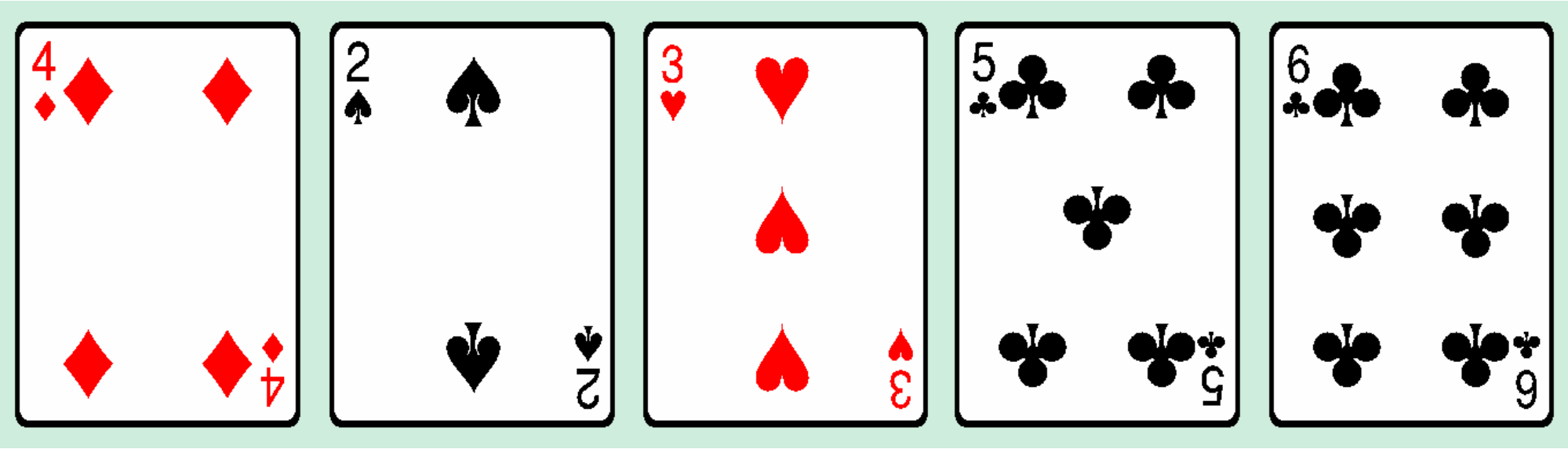
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



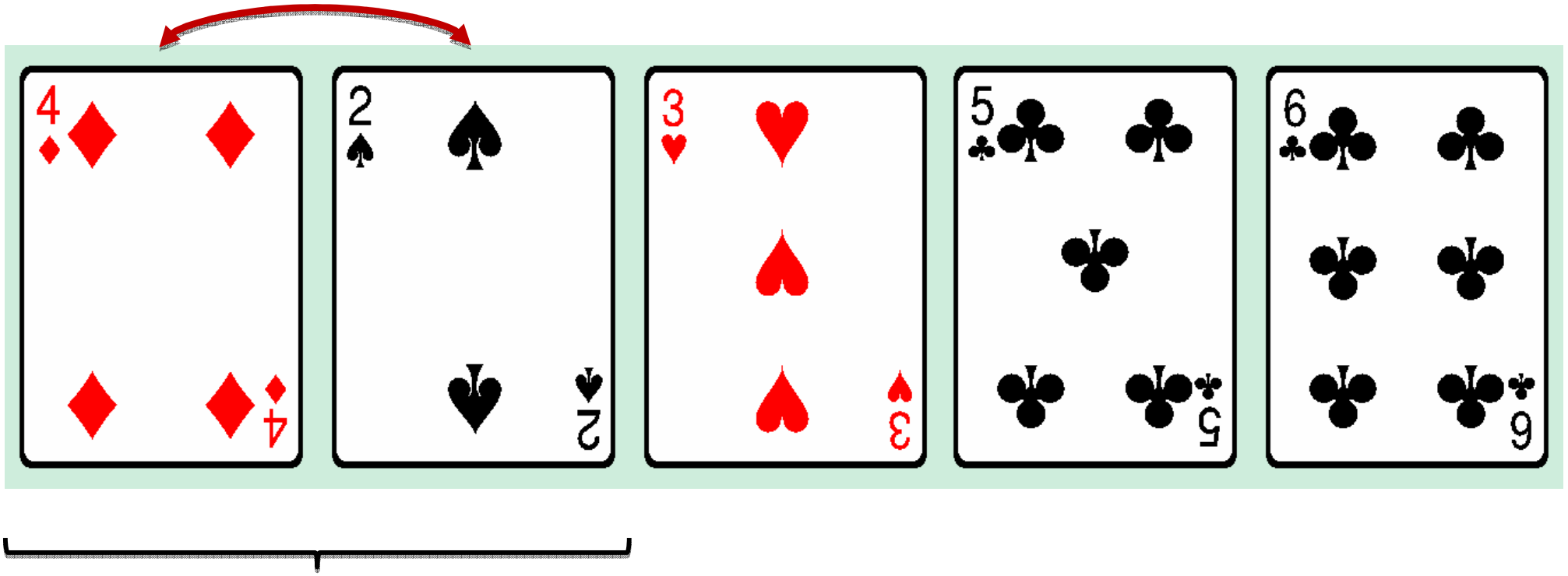
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



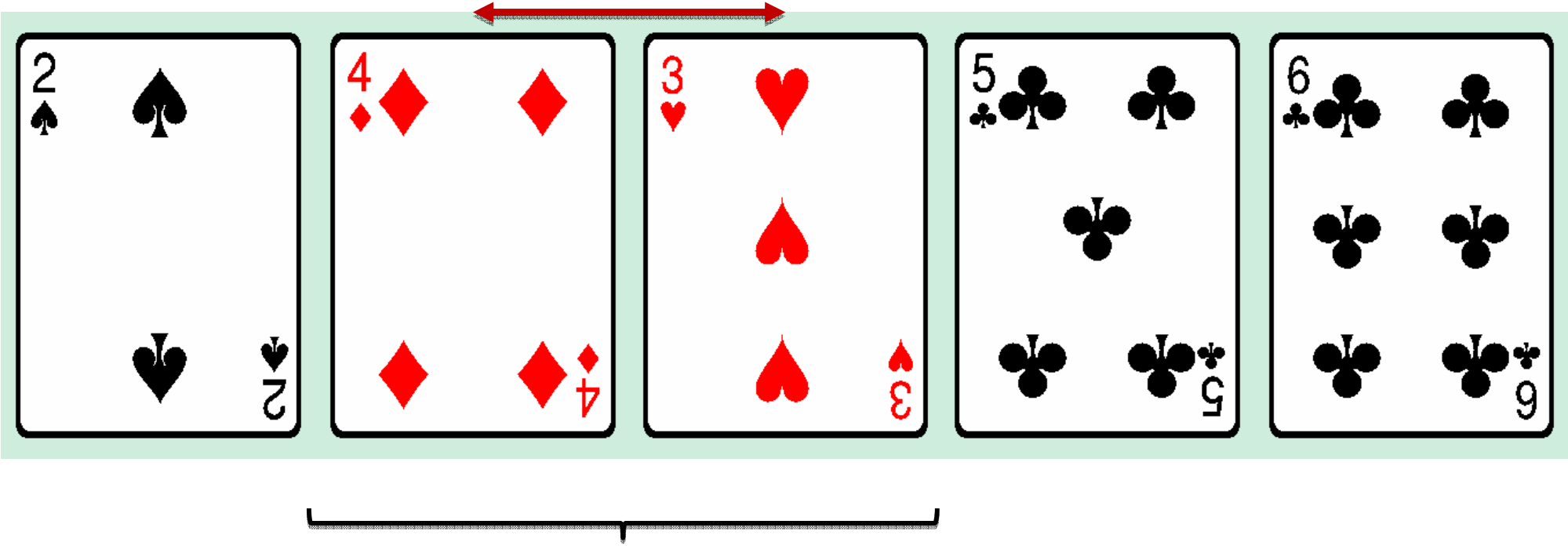
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



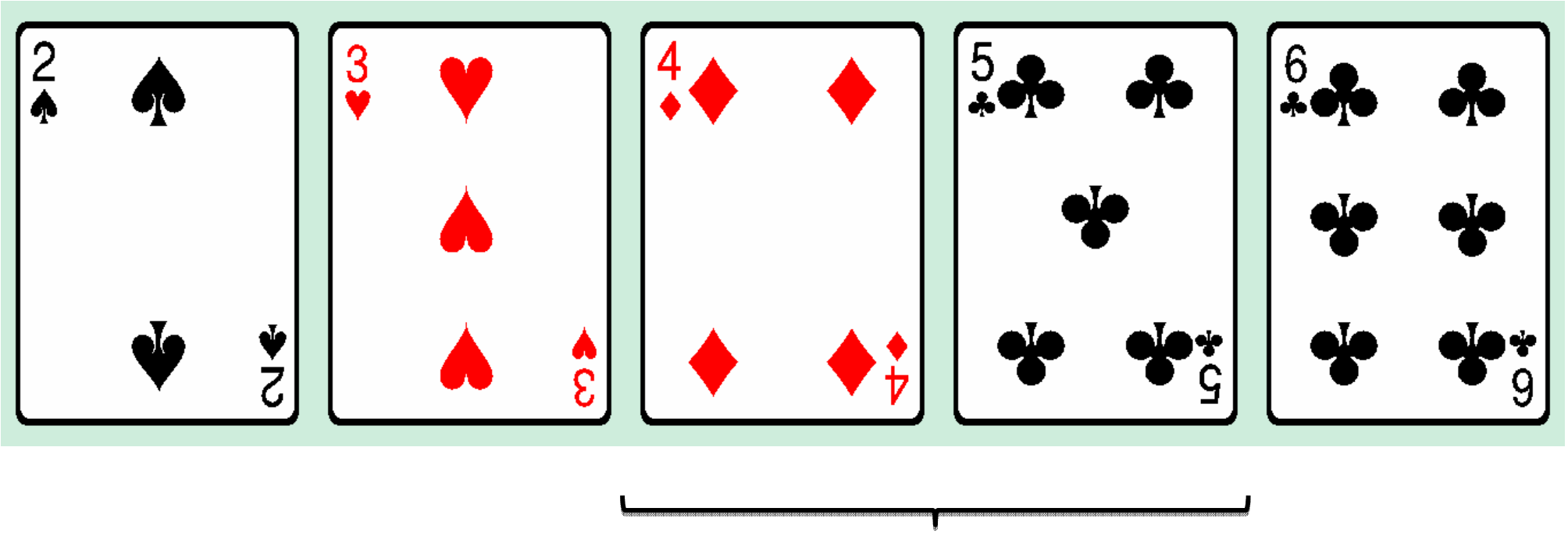
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



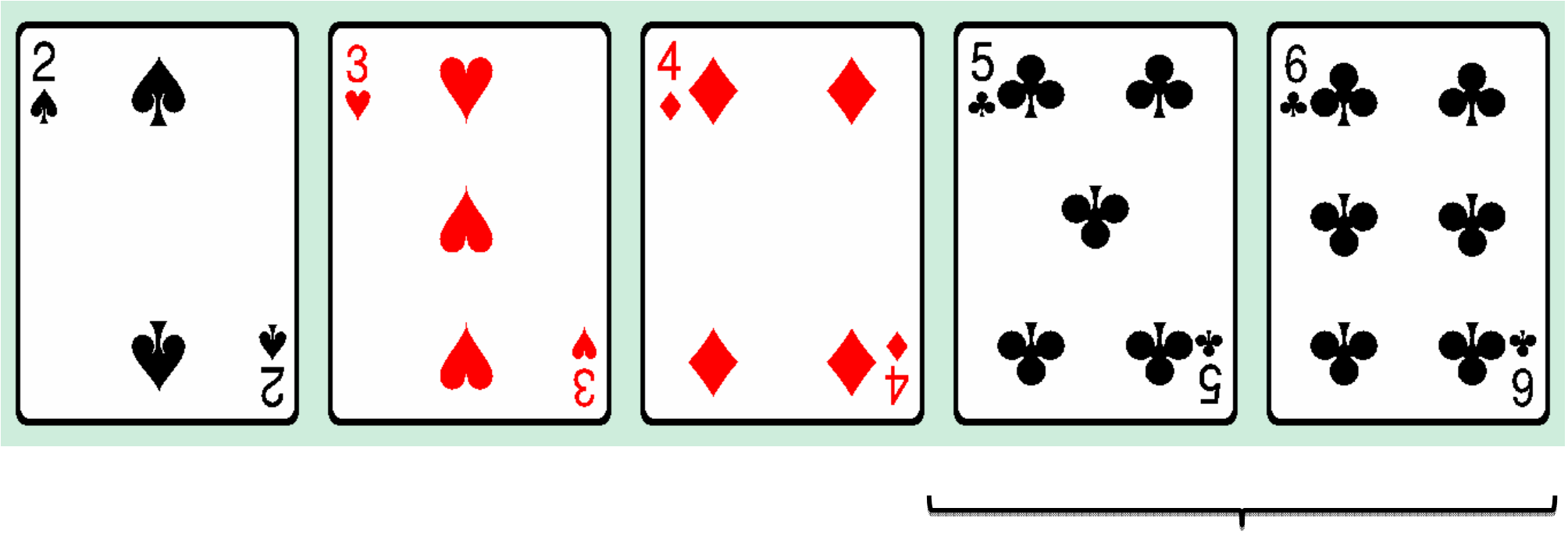
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



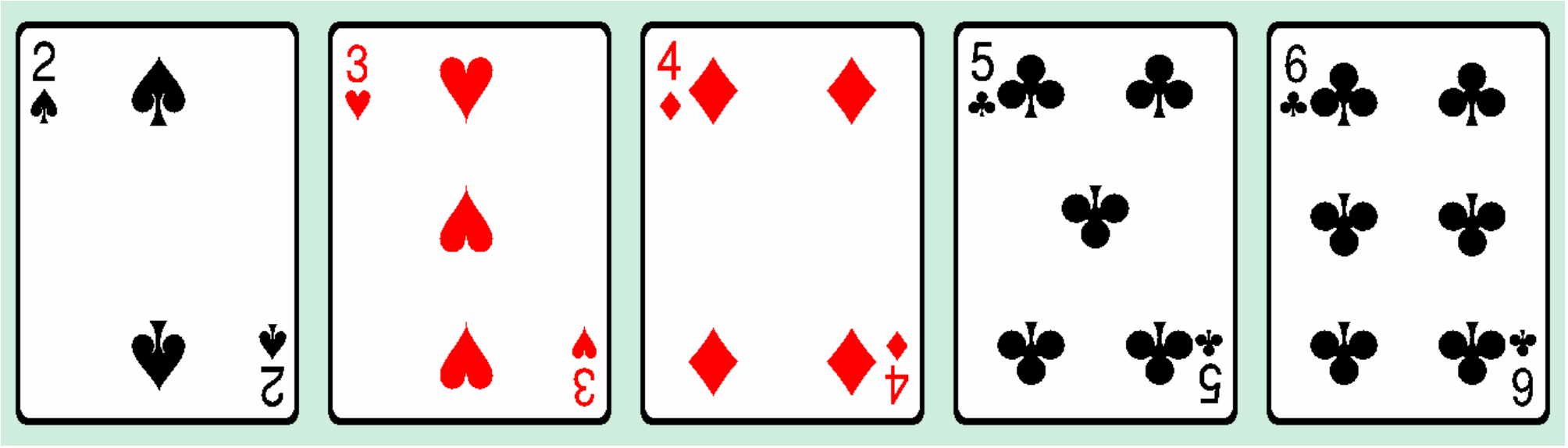
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



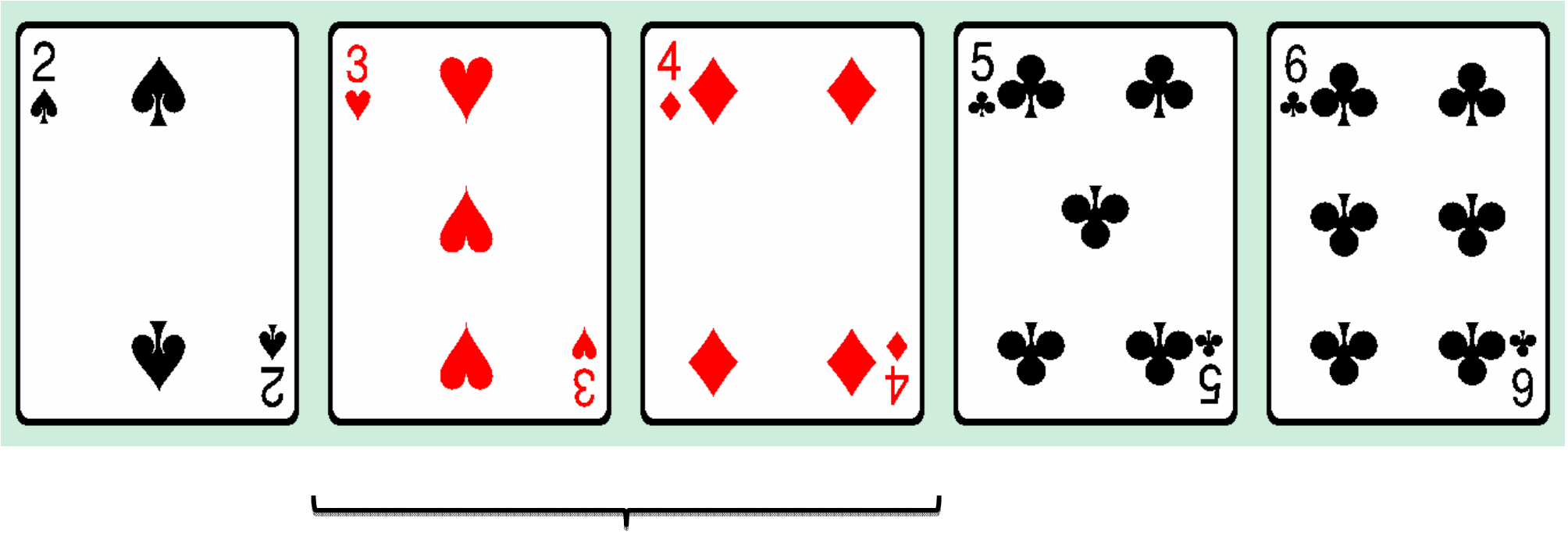
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



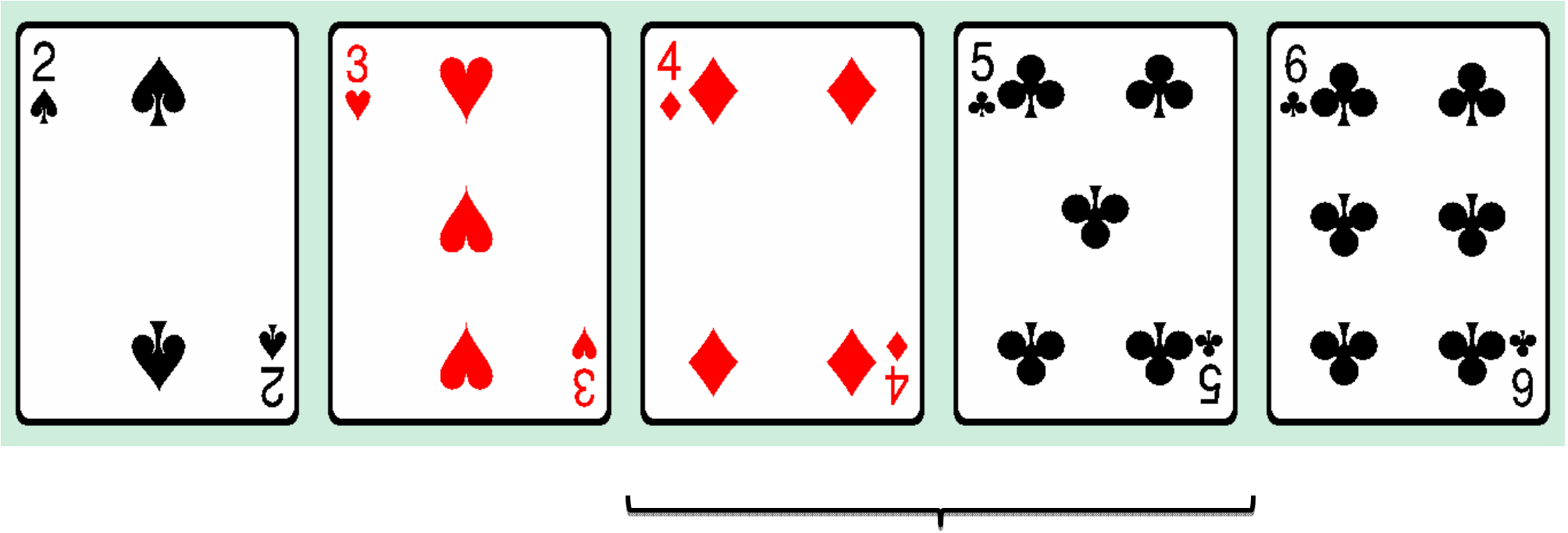
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



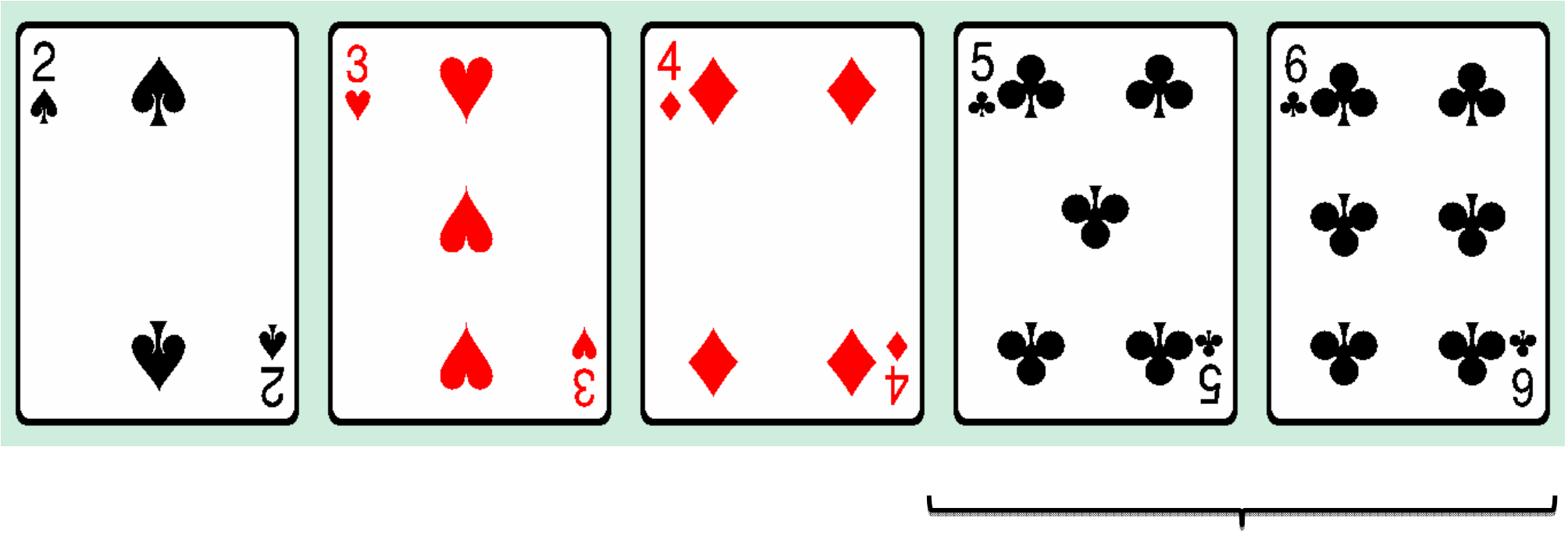
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



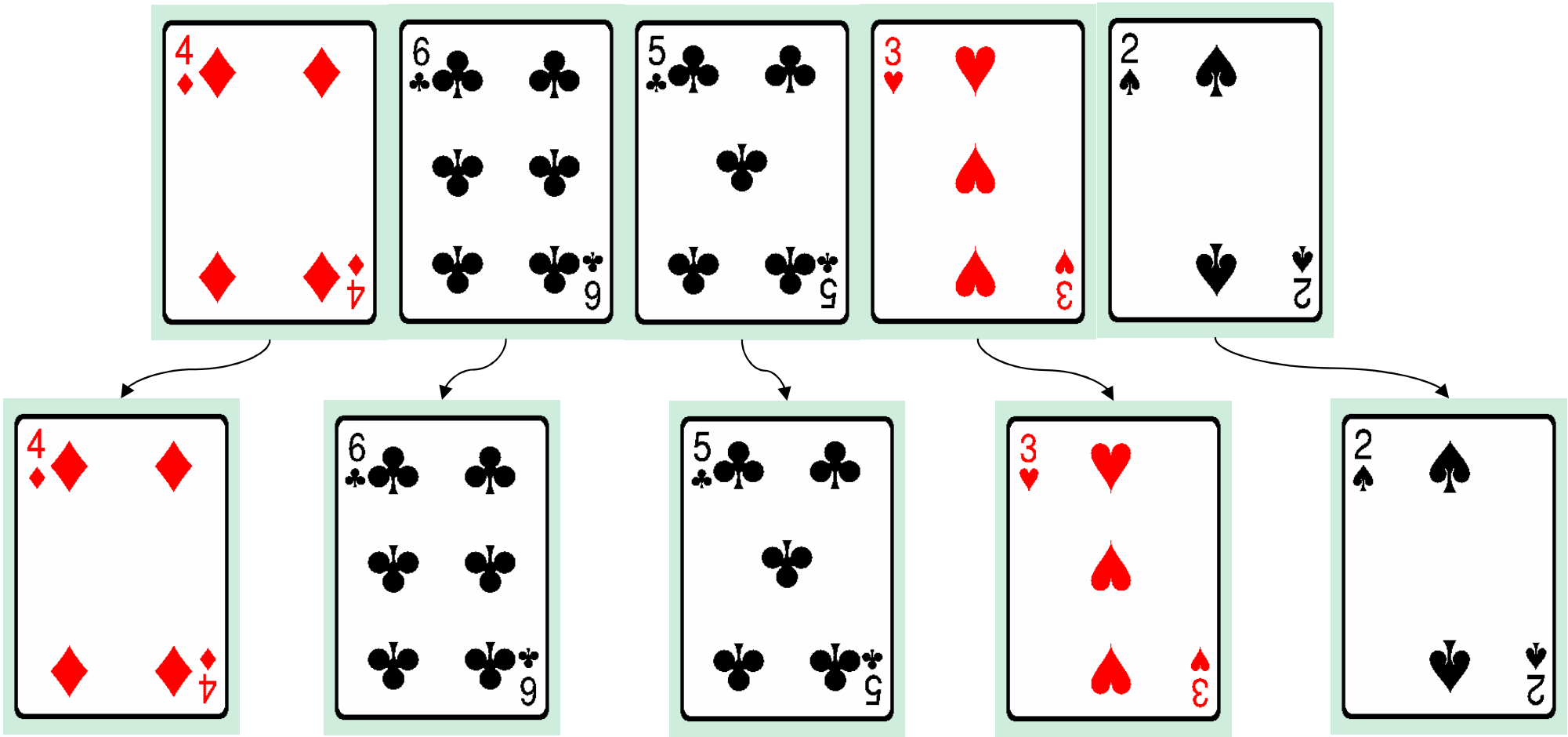
Základní strategie urychlování výpočtů

- ***BubbleSort*** má ***O(n²)*** ***kvadratická***



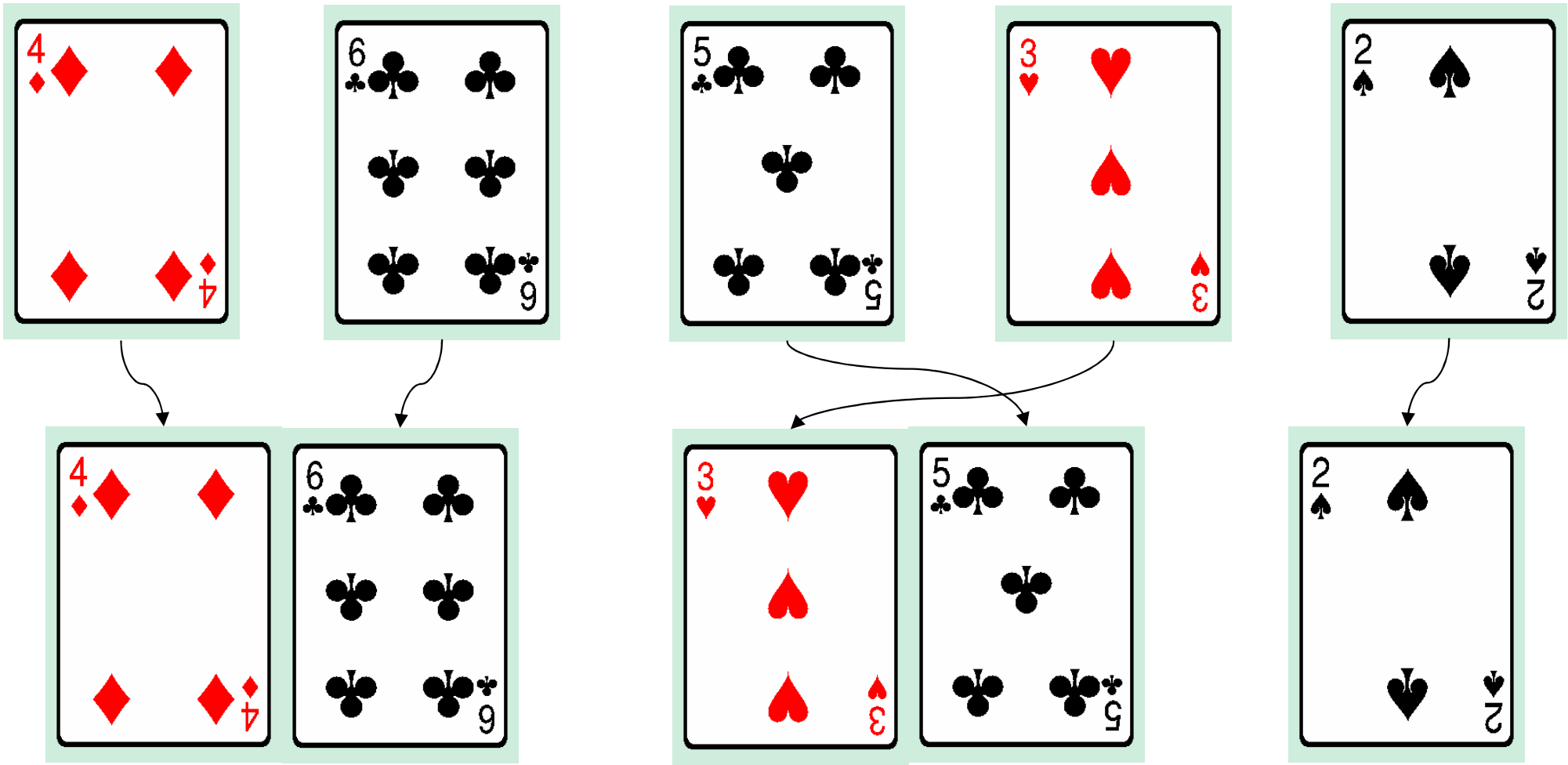
Základní strategie urychlování výpočtů

- ***MergeSort*** má ***O(n log n)*** ***linearitmická***



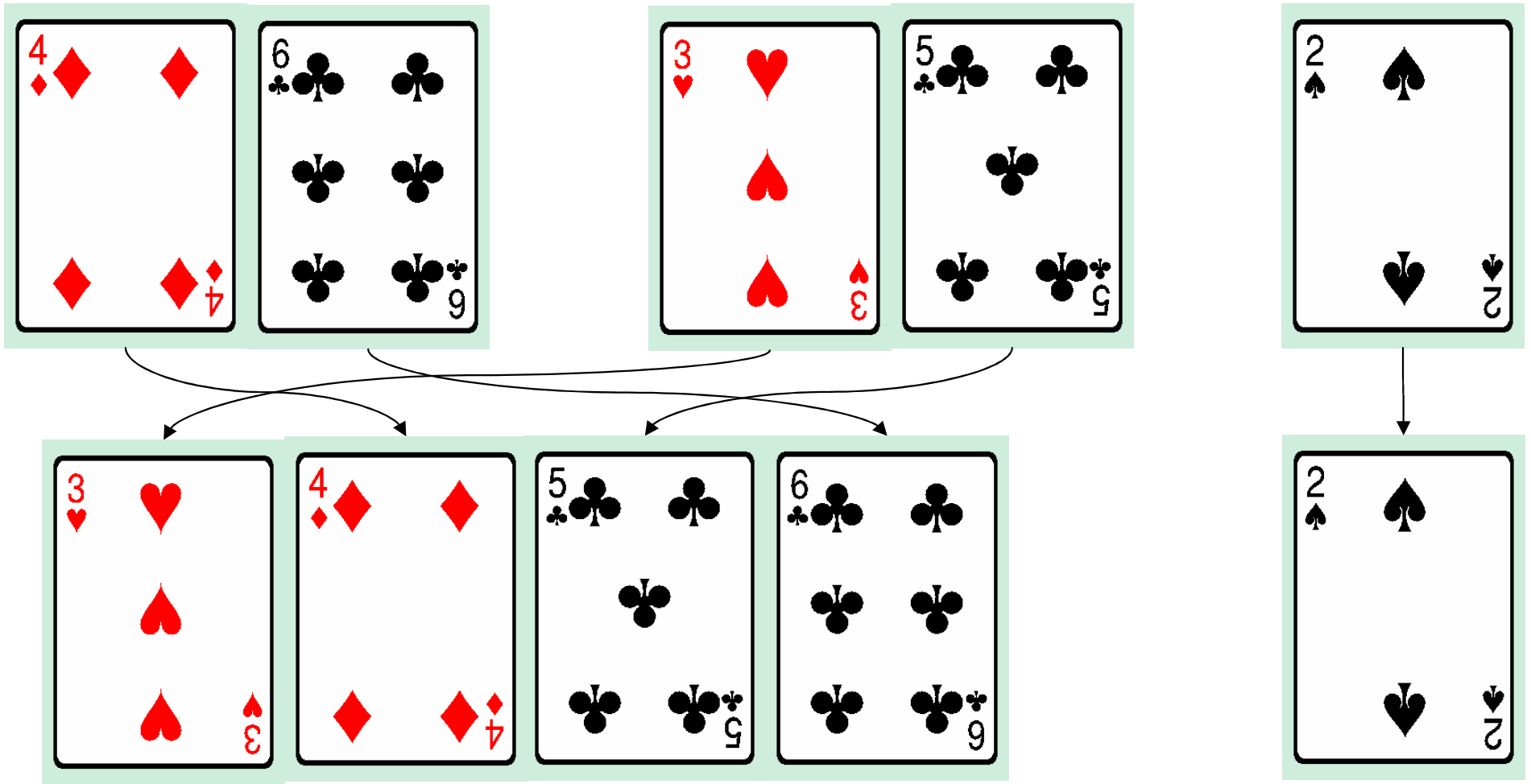
Základní strategie urychlování výpočtů

- ***MergeSort*** má ***O(n log n)*** ***linearitmická***



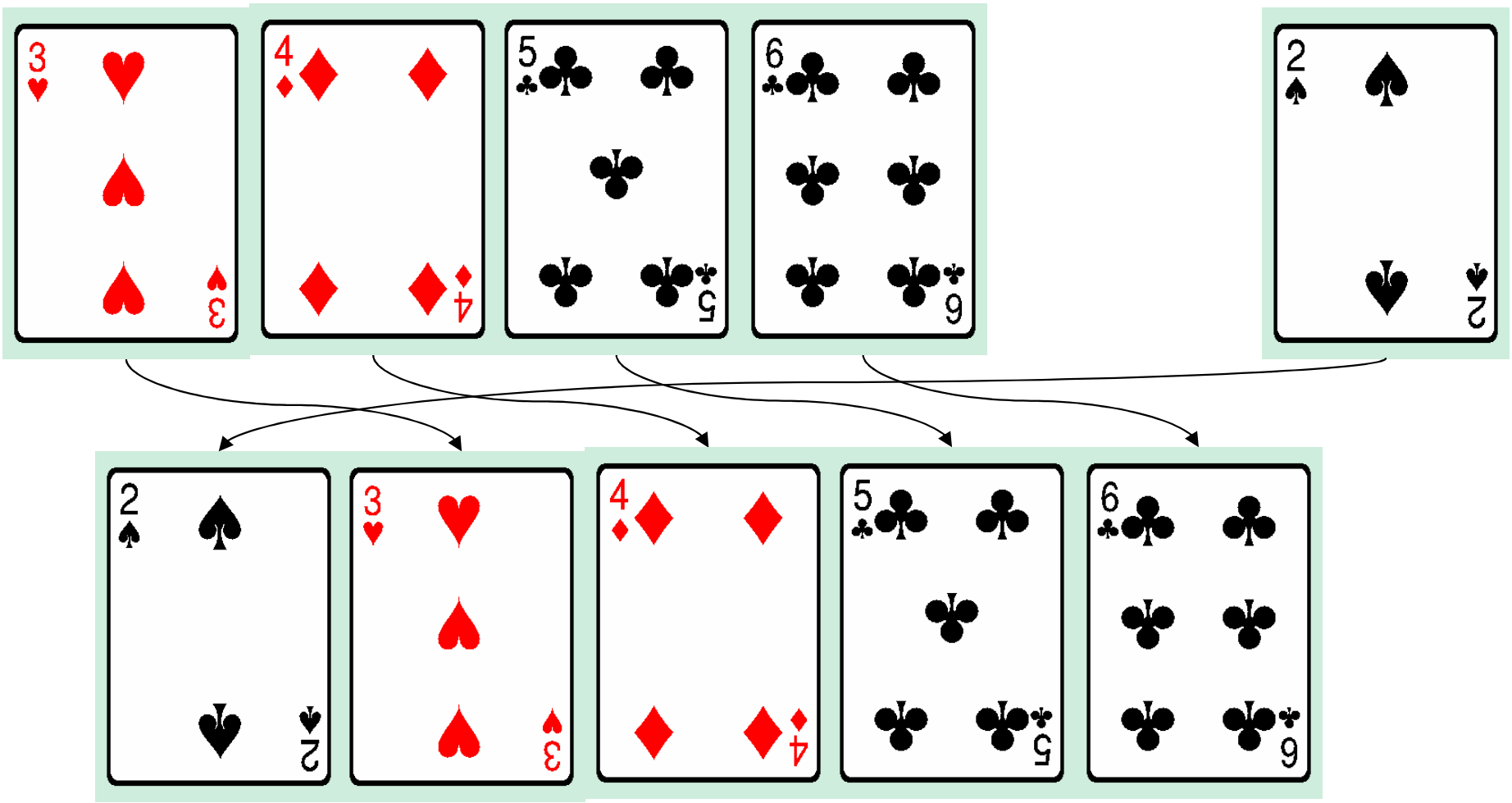
Základní strategie urychlování výpočtů

- ***MergeSort*** má ***O(n log n)*** ***linearitmická***



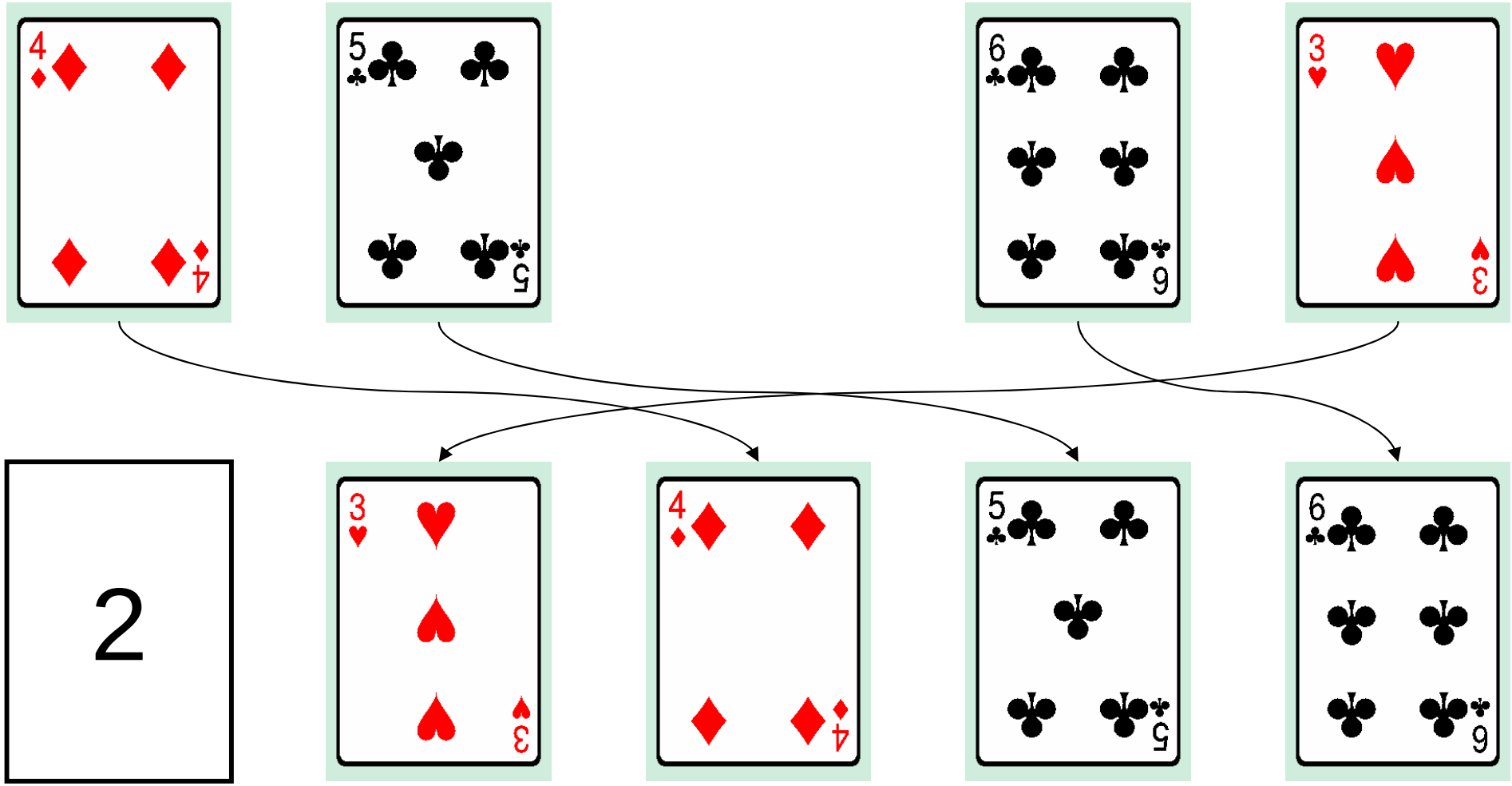
Základní strategie urychlování výpočtů

- ***MergeSort*** má ***O(n log n)*** ***linearitmická***



Základní strategie urychlování výpočtů

- ***RadixSort*** má ***O(kn)*** ***lineární***



Základní strategie urychlování výpočtů

- Optimalizace programu na „nízké úrovni“
  - Použití nižších programovacích jazyků, např. assembler

Základní strategie urychlování výpočtů

- Urychlování pomocí speciálního hardware
  - Použijeme zařízení, optimalizované pro danou operaci lépe než CPU
  - Matematický koprocesor (FPU – 80386, 80486)

  - FPGA (programovatelná pole)
  - Zvukové karty, grafické karty, PhysX, …

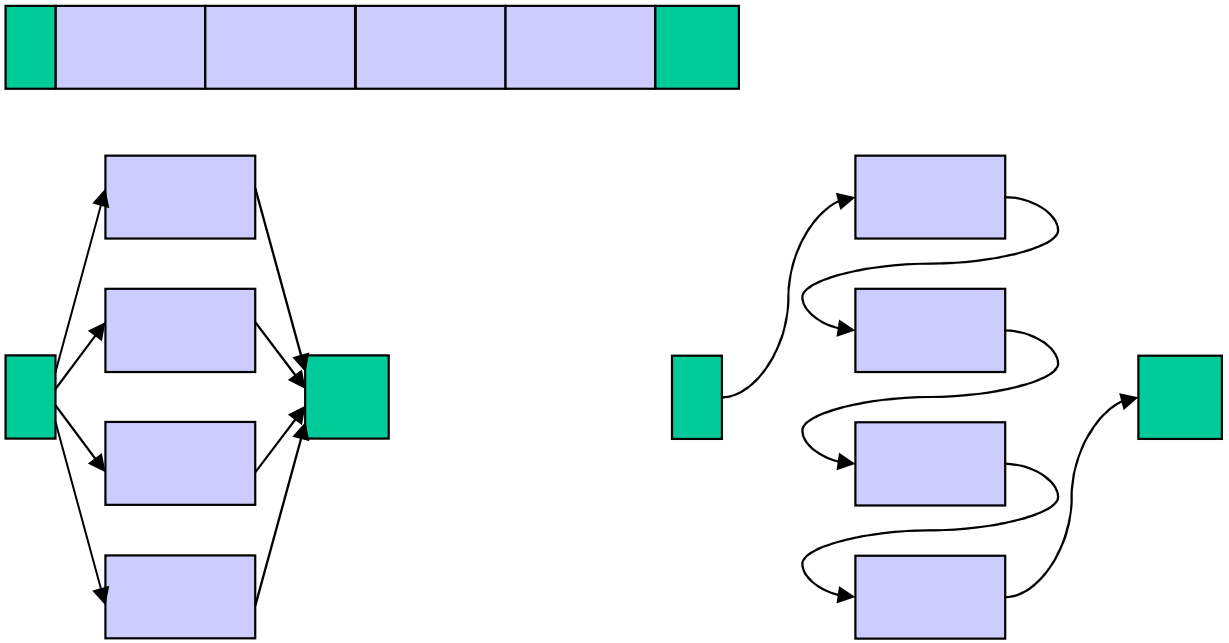
Shrnutí

- Program můžeme urychlit
  - Optimální algoritmy
  - Optimalizace na „nízké úrovni“ (assembler)
  - Speciální hardware

- Co když to nestačí?

Paralelizace

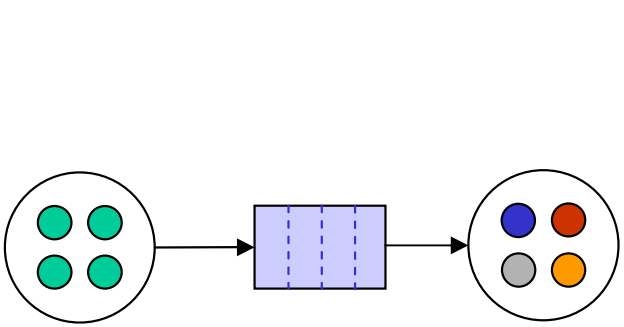
- Rozložení výpočtu na více celků
- Výpočet s každým celkem zvlášť

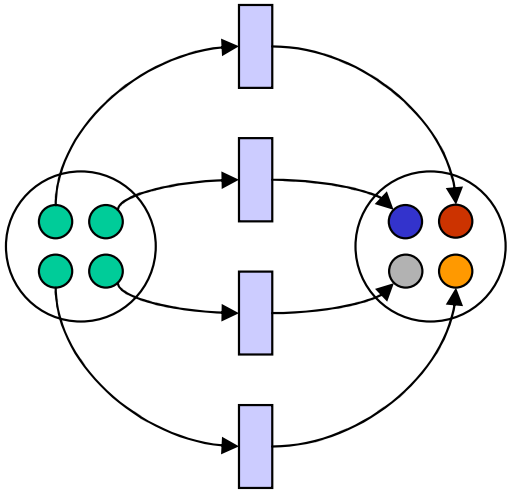


- Omezení
  - rozložitelnost výpočtu
  - počet paralelních ***vláken***

Paralelizmus

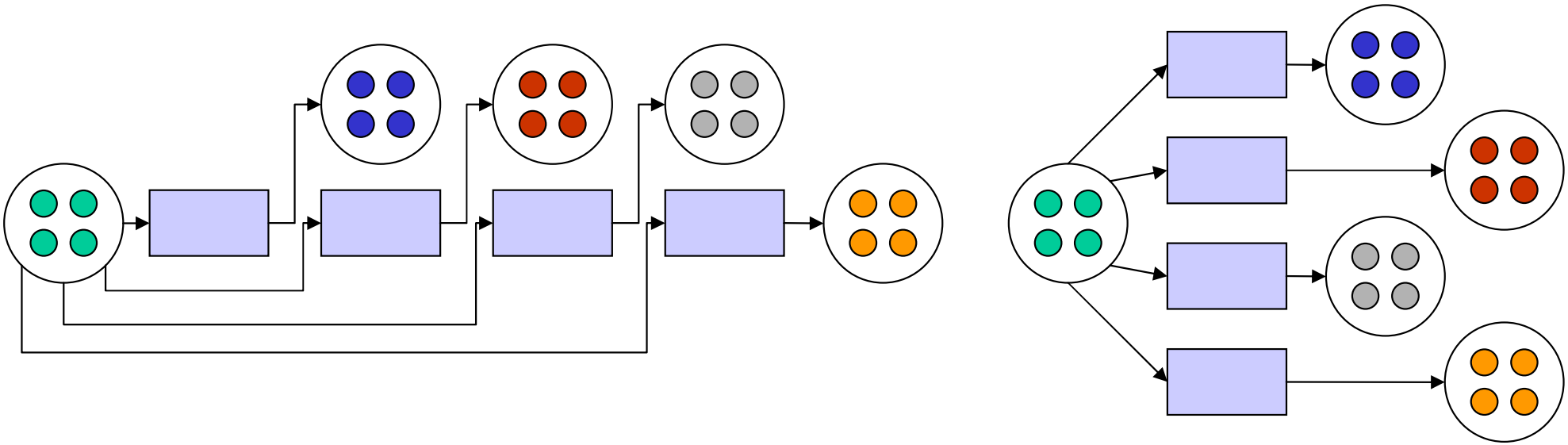
- Datový paralelizmus
  - Máme nějakou operaci, která se vykonává na velkém množství dat
  - Můžeme tedy rozdělit data a zpracovávat je ve více vláknech
  - Možnosti paralelizace závisí na objemu dat, tyto problémy lze typicky dobře škálovat (máme hodně práce i pro velké množství vláken)
  - Například sčítání dvou vektorů





Paralelizmus

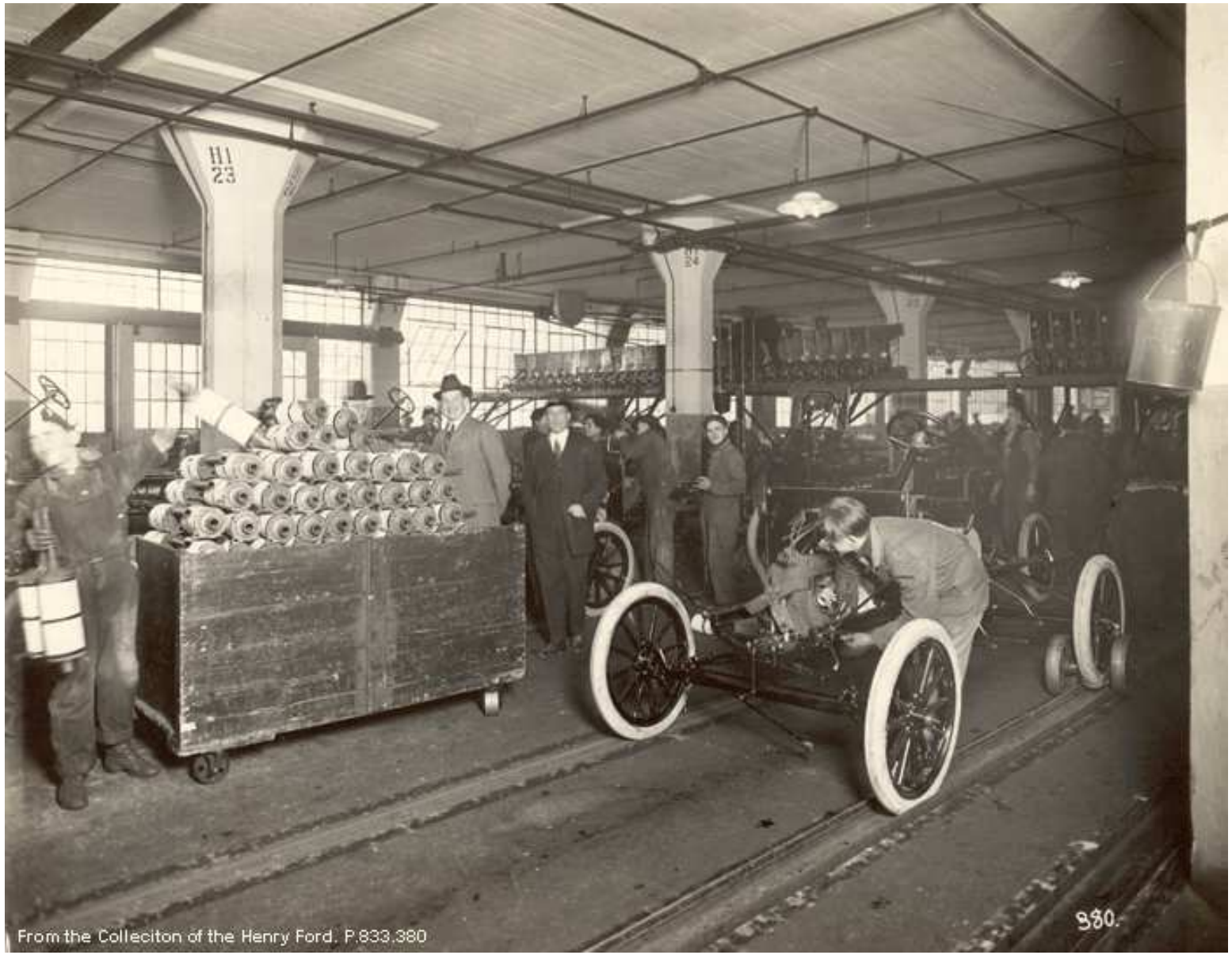
- Úlohový paralelizmus
  - Máme několik operací, které se vykonávají na nějakých datech (na těch samých, nebo na různých)
  - Můžeme tedy v každém vlákně vykonávat jinou úlohu
  - Obvykle máme jen omezený počet úloh, které chceme vykonat, pokud máme větší počet vláken, nemůžeme je využít



Paralelizace v reálném světě

Paralelizace v reálném světě

Paralelizace v reálném světě



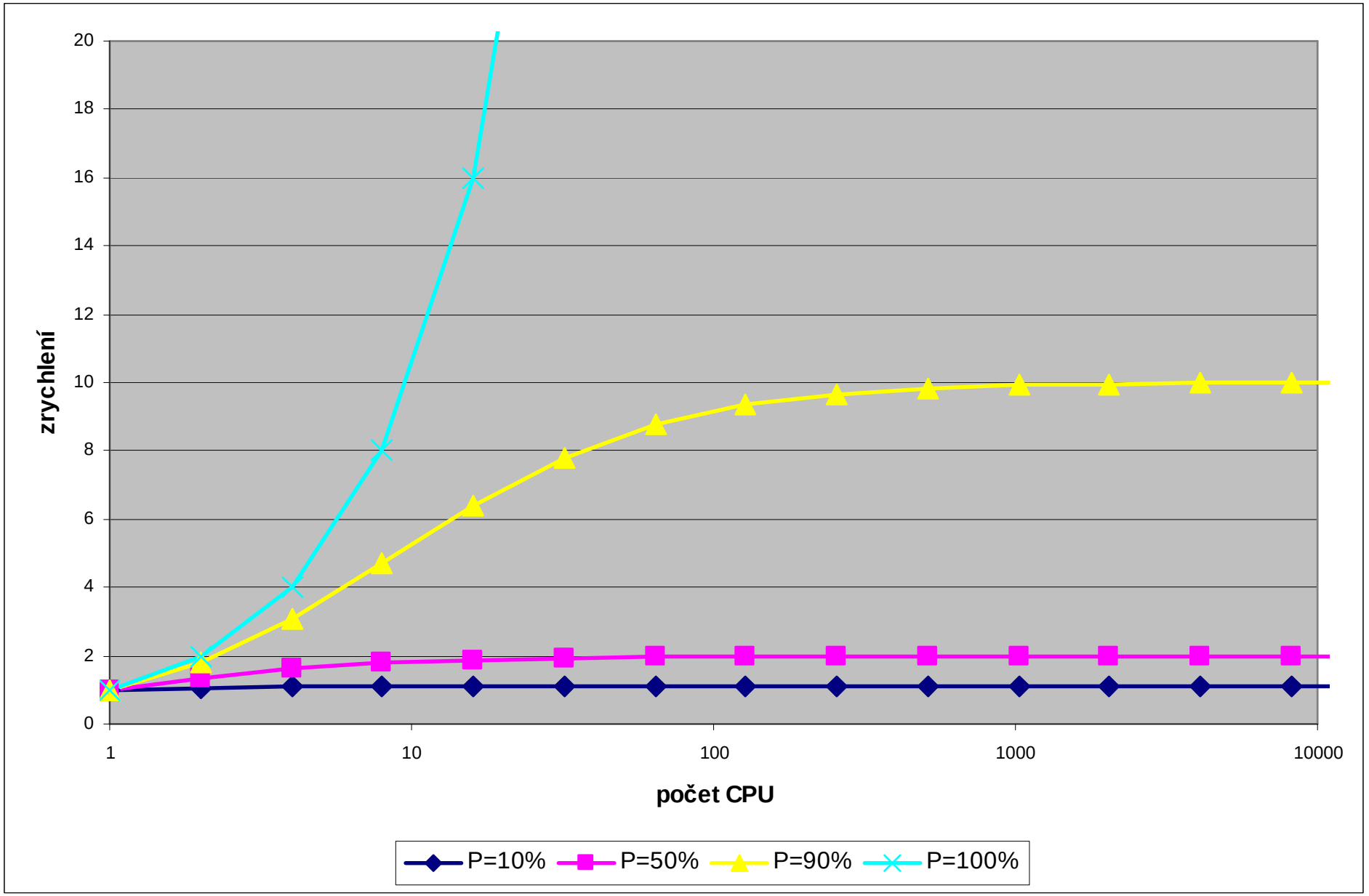
Paralelizace – docílené zrychlení

- Amdahlův zákon (Gene Amdahl)
- Ne všechny kroky algoritmu jsou vhodné pro paralelizaci
- Zabývá se limitem zrychlení na základě poměru paralelně vykonávaných částí k sekvenčně vykonávaným částem

$$zrychlení = \frac{1}{(1-P)+\frac{P}{N}}$$

- Kde ***P*** je část programu, běžící paralelně a ***N*** je počet vláken
- Aplikovatelné i na jinak než paralelně akcelerované programy

Amdahlův zákon



Amdahlův zákon

- Za jak dlouho N strávníků uvaří a sní jídlo z jednoho kotle?

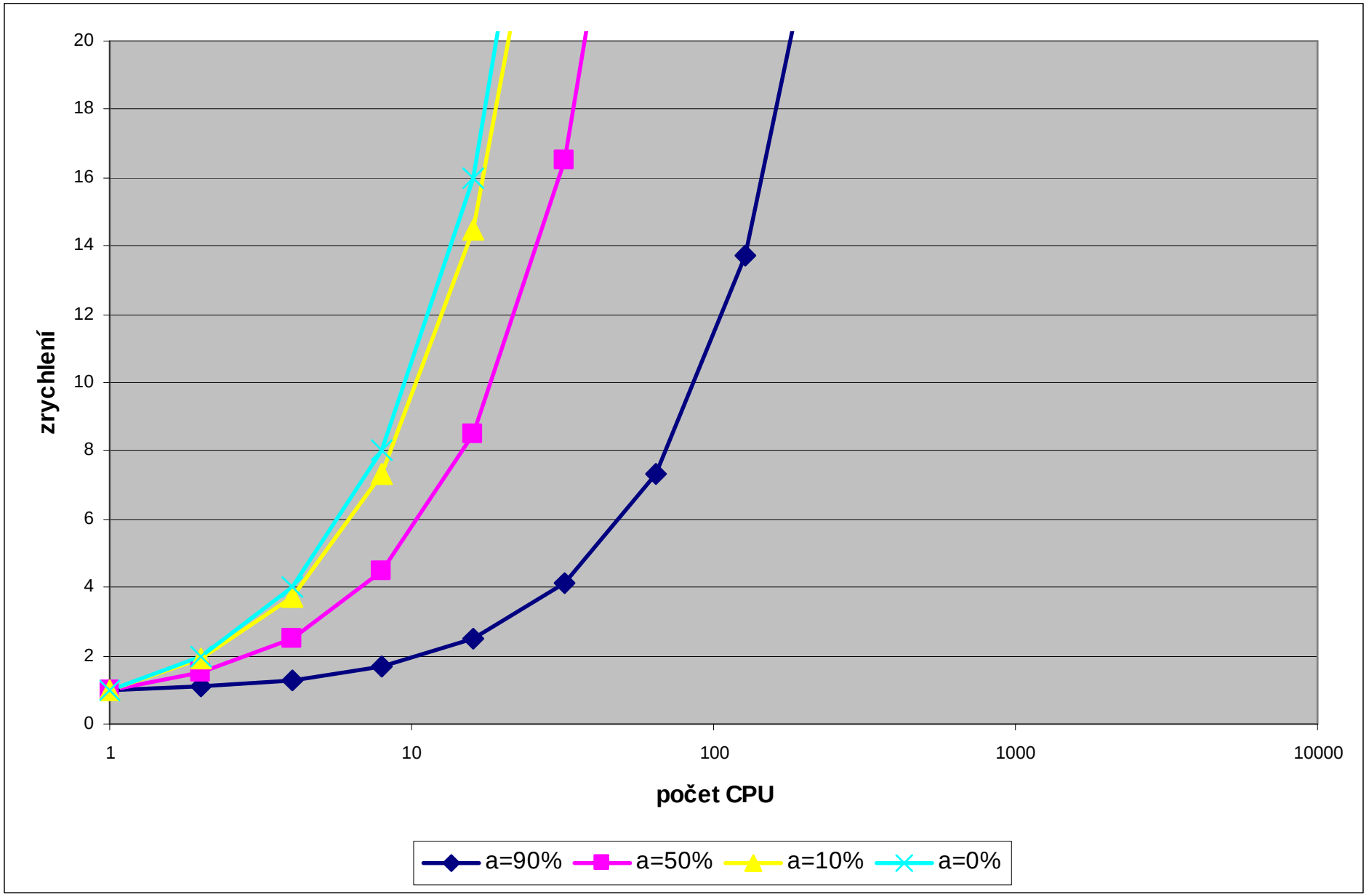
Paralelizace – docílené zrychlení

- Gustafsonův zákon (John L. Gustafson & Edward H. Barsis)
- Amdahlův zákon předpokládá pevnou velikost problému, zrychlení je tím pádem omezené limitou
- Gustafsonův zákon předpokládá že velikost problému bude růst s počtem vláken – zrychlení tedy není omezené

$$zrychlení = N - \alpha(N - 1)$$

- Kde α je část programu kterou **nelze** paralelizovat a ***N*** je počet vláken
- Omezení
  - Některé problémy nelze „natahovat“ donekonečna, Gustafsonův zákon potom nelze aplikovat

Gustafsonův zákon



Gustafsonův zákon

- Kolik stromů pokácí za den banda N dřevorubců?

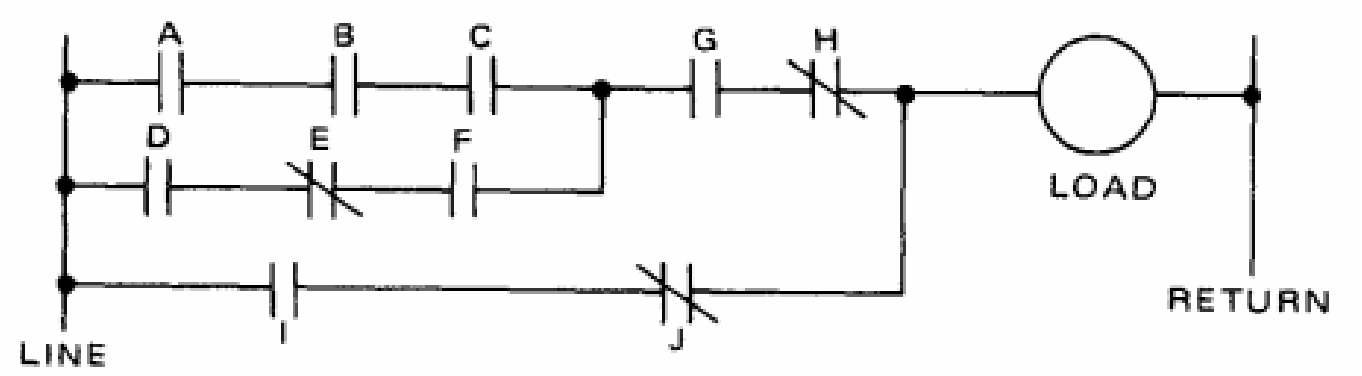
- (za předpokladu že s počtem dřevorubců přímo roste počet pokácených stromů)

Shrnutí

- Víme co je to paralelizace a jaké jsou její základní formy
- Víme jaké zrychlení můžeme očekávat
  - Amdahlův zákon
  - Gustafsonův zákon

Paralelizace v praxi: Šířka slova

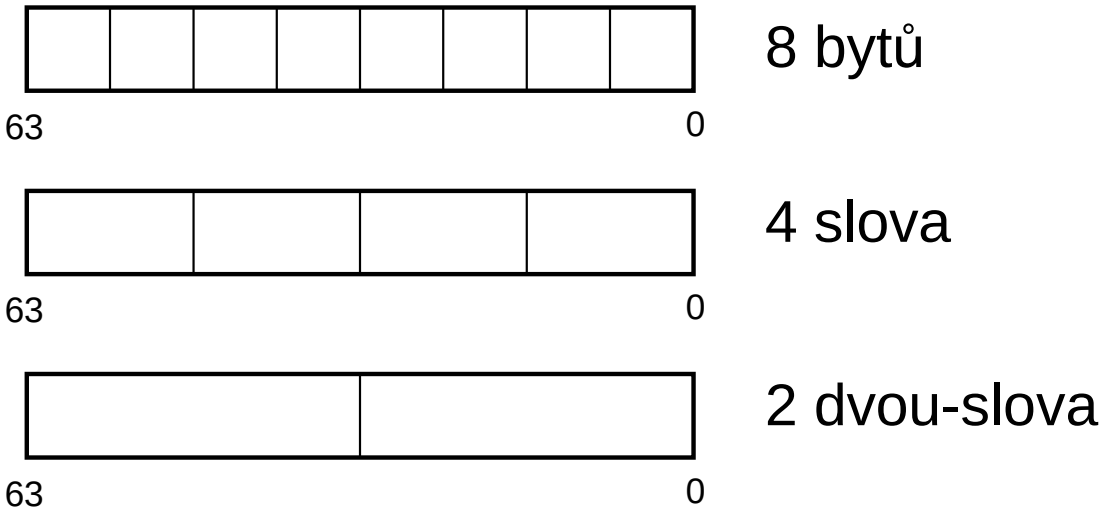
- Počet bitů lze chápat jako jednoduchý paralelismus
- Motorola MC14500B
  - Jednobitový procesor
  - K řešení tzv. „ladder diagrams“



- Proti tomu dnešní procesory mají 64 bitů
  - Např. při sčítání dvou čísel 64x rychlejší (pokud pomineme taktovací frekvenci)

Paralelizace v praxi: SIMD instrukční sady

- Tzv. „multimediální“ instrukční sady
  - MMX, 3DNow!, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4a, SSE5, AVX
- Nové registry, reprezentující více než jednu hodnotu
  - Např. MMX registry:



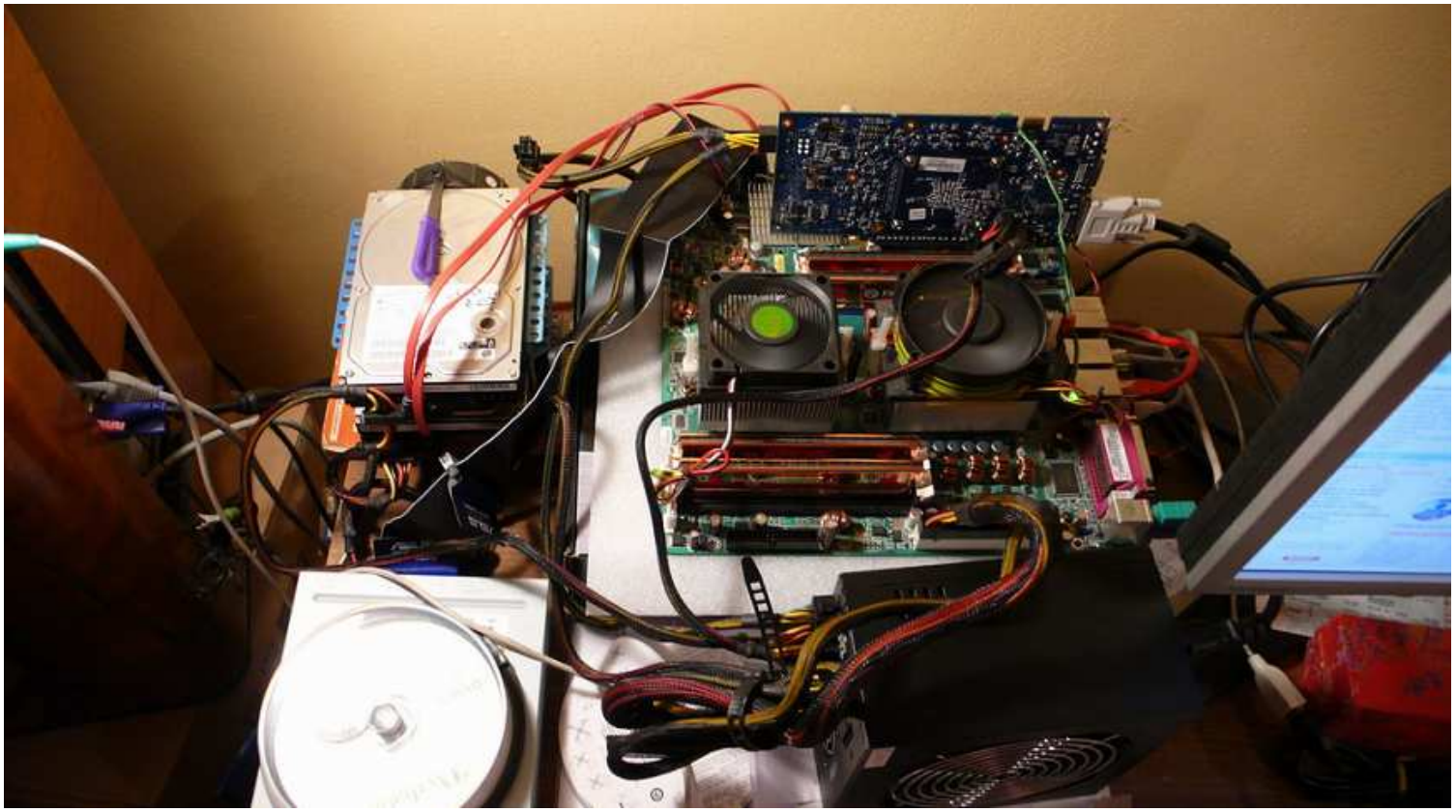
- Lze provádět (stejné) operace s více hodnotami najednou

Další úrovně paralelizmu

- Vlákna
  - Primitiva operačního systému
  - Části jednoho ***procesu***, sdílejí paměťový prostor
- Procesy
  - Primitiva operačního systému
  - Oddělený paměťový prostor, možnost využít ***sdílenou paměť***
  - Vytvoření procesu je pomalejší než vytvoření vlákna
- Počítače
  - Na různých počítačích mohou běžet spolupracující procesy
  - Velmi oddělený paměťový prostor
  - Komunikace zpravidla po síti

Paralelní architektury - CPU

- Moderní CPU mají zpravidla více ***jader***
- Existují základní desky na které lze osadit více CPU
- Dříve také využití technologie ***hyperthreading***
  - Dnes se spíše upouští, neefektivní, neekologické



Paralelní architektury - GPU

- Grafické karty, Graphics Processing Units
- Čím dál častěji využívané pro vědecké výpočty
- Výpočetní výkon v řádu TFLOPS, 10x až 20x více než CPU
- Paralelní architektura s velkým množstvím jader (stovky)
- GPU jsou schopné paralelně vykonávat tisíce vláken přičemž naplánovaných vláken je ještě mnohem více
- Dříve pouze programování za pomoci grafických knihoven (OpenGL, DirectX)
- Dnes existují specializované jazyky (OpenCL, CUDA, CTM)
- Na budoucích architekturách (Intel Larrabee) by měl běžet i „normální“ x86 kód.

Paralelní architektury - Clustery

- Cluster je skupina počítačů spojená sítí (LAN)
- Počítače přitom úzce spolupracují
- Cílem je buď zvýšit dostupnost služeb (servery)
  - HA (high availability clustery)
  - Často jen dva počítače
- Nebo poskytnout vyšší výkon
  - Load-balancing clusters, výpočetní clustery
  - Pro uživatele se chovají jako jeden počítač
  - Programují se pomocí speciálního API
    - např. (Open) MPI
  - Často clustery X-Boxů nebo PSX

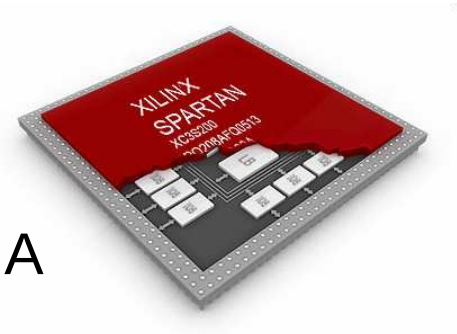
Paralelní architektury - Grid

- Velmi podobné jako clustery
- Jednotlivé úlohy spolu však typicky nekomunikují
  - Jednoduše paralelizovatelné úlohy
  - Např. řešení problémů přes internet (SETI@home)
- Často se také používá k řešení více úloh najednou
- Může mít (geograficky) větší rozsah než clustery
- „Netváří se“ jako jeden počítač, k rozdělení úloh mezi počítače používá speciální software

Paralelní architektury - Masivně paralelní arch.

- MPP – masivně paralelní procesor
- Počítače s velkým množstvím procesorů propojených sítí
- Na rozdíl od clusterů se používá speciální síť (ne LAN)
- Typicky tisíce – statisíce procesorů
- Každý procesor má kopii OS a běžící aplikace

Paralelní architektury - FPGA / ASIC



- FPGA (field programmable gate array)
  - Programovatelné hradlové pole
  - Lze softwarově nastavit propojení obvodů uvnitř FPGA
  - Specifikace obvodu ve speciálním jazyce (VHDL)
  - Takto se dají vytvořit optimalizované obvody, paralelně zpracovávající daný problém
  - Nebo třeba klasický procesor (PicoBlaze)
  - Výhodou je nízká cena, rychlý vývoj

- ASIC (Application-specific integrated circuit)
  - Je tzv. „zákaznický obvod“, vyrobený na míru pro danou aplikaci
  - Nevýhodou je vysoká cena za výrobu litografické masky

Shrnutí

- Víme jaké máme hardwarové / softwarové úrovně paralelismu
- Víme jaké máme k dispozici architektury

Paralelní programování

- Musíme se vypořádat s několika problémy
  - Příprava (distribuce) dat a sběr výsledků
  - Synchronizace a závislosti obecně
  - Balancování zátěže

Distribuce dat v paralelním programu

- V rámci jednoho procesu
  - Paměťový prostor je sdílený, což je pro některé algoritmy výhodné
  - Omezení paměti na 2GB (na starých 32 bit architekturách)
  - Typicky žádná distribuce, vlákna jednoduše objekty v paměti sdílí
- Jinak ...
  - Použití sdílené paměti
  - Rozesílání dat po síti
  - Nebo vůbec žádná společná data

Datové závislosti

- Některé algoritmy nemají datové závislosti, např.:
  - Zobrazovací algoritmy (raytracing)
  - Hledání řešení úloh hrubou silou (kryptografie)
- Sekvenční závislost – jeden krok algoritmu produkuje výsledky pro další
- Anti-závislost – další krok algoritmu by přepsal výsledky prvního
- Výstupní závislost – dva kroky algoritmu zapisují na stejné místo, výsledky tedy musí zapsat poslední krok

Balancování zátěže

Paralelní programování

- Pokud nemáme zdrojové kódy programu
  - Pokud program není sám o sobě paralelní, máme smůlu
  - Někdy ale můžeme program spustit víckrát naráz, pokud nám to v něčem pomůže
    - Např. zvětšování fotek
    - Využití příkazového řádku

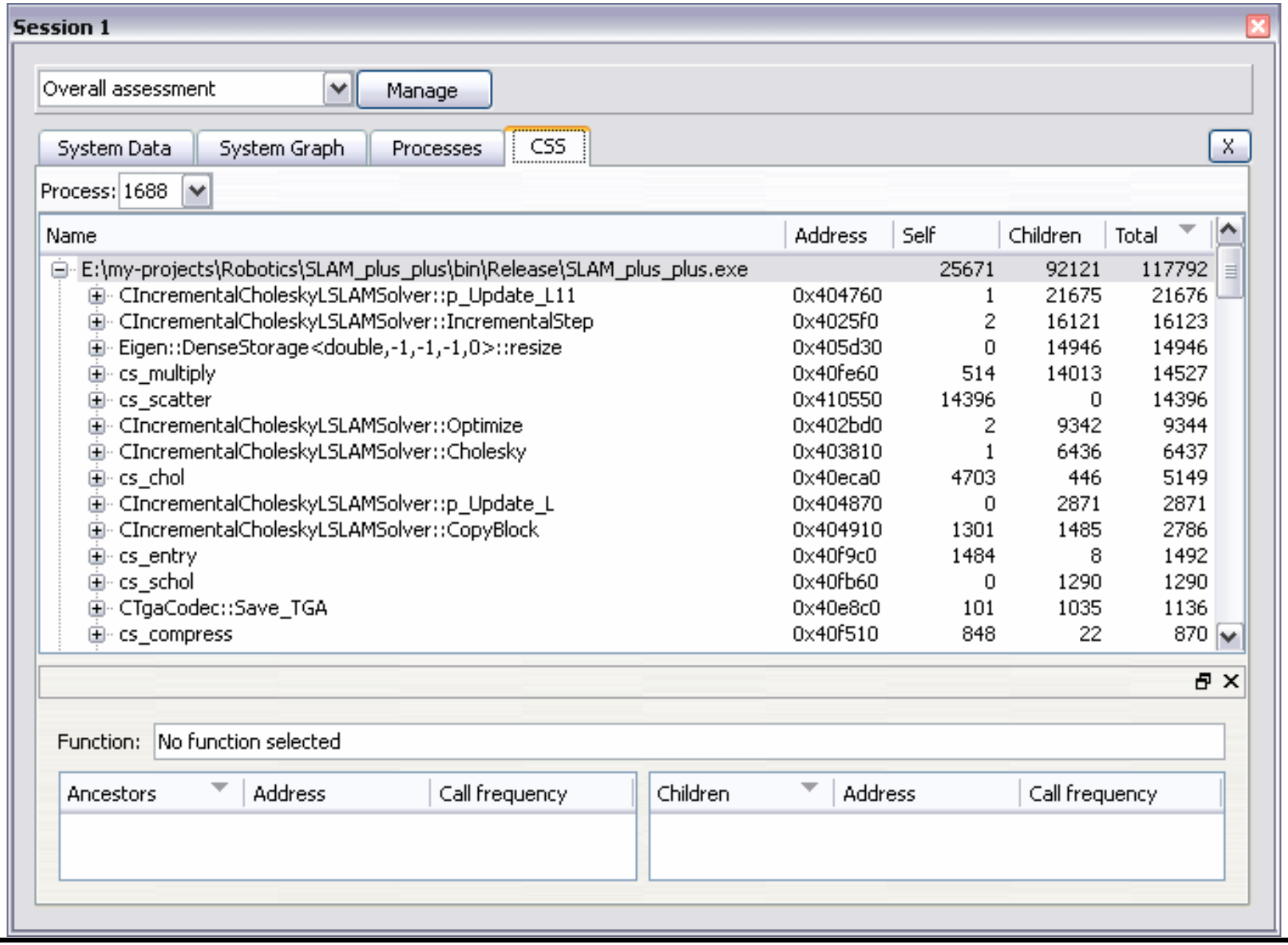
Paralelní programování

- Dema na: http://www.fit.vutbr.cz/~ipolok/ivs/dema.zip
- <screencast>

Paralelní programování

- Pokud máme zdrojové kódy programu
  - Většinou neparalelizujeme celý program, ale jen jeho část jejíž vykonání zabere nejvíc času
    - Profiling
  - Můžeme buďto využít specializovanou knihovnu
    - Vysoká úroveň
  - Nebo napsat paralelní část programu „ručně“
    - Nízká úroveň

Ukázka: Profiling



Integrated Performance Primitives

- Knihovna hotových funkcí jež běží paralelně
- Načítání / ukládání multimédií
- Počítačové vidění
- Kryptografie
- Komprese
- Zpracování obrazu / signálů
- Raytracing / rasterizace
- Rozpoznávání řeči

AMD Performance Library

- Framewave
  - Multimedální funkce jako IPP
- AMD Core Math Library
  - Základní lineární algebra (BLAS)
  - Další lineární algebra (LAPACK, FFT)

NVIDIA

- Nabízí podobné knihovny jako Intel nebo AMD
- Akcelerace pomocí GPU a CUDA
- CUBLAS
- CULA (LAPACK)
- CUFFT

API pro „nízké“ paralelní programování

- Můžeme použít systémové API (linux / pthreads / winapi)
  - Máme poměrně dobrou kontrolu nad programem
  - Ale může to být nepřehledné, kód je těžko udržovatelný / rozšiřitelný
- Některé platformy se programují pomocí speciálního API
  - např. GPU se programují v GLSL, CUDA nebo OpenCL
- Můžeme použít některou ze specializovaných knihoven
  - MPI
  - OpenMP

Message Passing Interface

- Používá se na clusterech a superpočítačích
- Základní jednotkou je proces
- MPI vytvoří potřebný počet instancí procesu a zprostředkovává výměnu zpráv mezi procesy
  - Jednak mezi dvěma procesy (point-to-point)
  - Jednak globální (broadcasting)
- Všechny procesy vykonávají ten samý kód, odlišují se jen podle svého MPI id

Message Passing Interface

```
#include <mpi.h>

int main(int argc, char *argv[])
{
   int numprocs, myid; // kolik je procesů a id tohoto procesu
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &myid);

   if(myid == 0) {
       // toto je „první proces“, většinou koordinuje ostatní,
       // stará se o zasílání práce a sestavení výsledků
   } else {
       // toto jsou všechny ostatní, vykonávají práci
   }

   MPI_Finalize();
   return 0;
}
```

Message Passing Interface

- Vyzkoušejte si demo na PCIVS
  - <stažení a překlad dema>

Open Multi Processing

- Používá model sdílené paměti – pouze na jednom počítači
- Rozšíření i pro distribuované systémy
- Podpora mnoha platforem (včetně Windows)
- Sestává ze sady direktiv kompilátoru (C / C++ / Fortran)
- Direktivy označují bloky kódu, jež má běžet paralelně
- V každém bloku mohou vlákna zjistit svoje id a jejich počet
- Obsahuje direktivy pro synchronizaci

Open Multi Processing

```
#include <omp.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
   int myid, numthreads;
   #pragma omp parallel private(myid) // zde se vytvoří vlákna
   {
        myid = omp_get_thread_num();
        printf("Hello World from thread %d\n", myid);
        #pragma omp barrier // zde se vlákna synchronizují
        if(myid == 0) { // pouze první vlákno
               numthreads = omp_get_num_threads();
               printf("There are %d threads\n", numthreads);
        }
   }

   return 0;
}
```

Open Multi Processing

- Vyzkoušejte si demo na PCIVS
  - <stažení a překlad dema>

CUDA, OpenCL

- Program běží na CPU (***host***) a GPU (***device***)
- CPU řídí provádění kódu a volá funkce na GPU (***kernels***)
- CUDA
  - Proprietární API od NVIDIA
  - Existuje jako ***Runtime*** a ***Driver API***
  - ***Runtime*** funguje jako nadstavba kompilátoru, kde se ve stejném zdrojovém souboru vyskytuje kód pro CPU i pro GPU
  - ***Driver API*** vyžaduje zkompilovaný program v CUDA (.ptx / .cubin), obsahuje funkce pro jeho nahrání do GPU a následně volání
  - K dispozici zdarma knihovny CUBLAS a CUFFT
- OpenCL
  - Mladší API, původně od Apple, nyní vyvíjeno Khronos group
  - Oproti CUDA podporuje i jiné zařízení než GPU
  - Funguje jako ***Driver API,*** OpenCL programy se při spuštění aplikace kompilují ze zdrojového kódu (možnost uložení binární podoby)

CUDA Runtime

```
__global__ void VecAdd(const int *A, const int *B,
                       int *C, int N) // kernel - kód pro GPU
{
   int myid = blockDim.x * blockIdx.x + threadIdx.x;
   // zjistí id vlákna

   if(myid < N)
       C[myid] = A[myid] + B[myid];
   // provede přičtení jednoho elementu
}
```

CUDA Runtime (pokračování)

```
int main(int argc, char *argv[]) // kód pro CPU
{
   int N = ..., size = N * sizeof(int); // velikost problému
   int *hA=new int[N],*hB=new int[N],*hC=new int[N]; // data CPU
   int *dA,*dB,*dC;
   cudaMalloc((void**)&dA, size);
   cudaMalloc((void**)&dB, size);
   cudaMalloc((void**)&dC, size); // naalokuje data na GPU
   cudaMemcpy(dA, hA, size, cudaMemcpyHostToDevice); // kopie
   cudaMemcpy(dB, hB, size, cudaMemcpyHostToDevice); // na GPU
   int WGsize = 512; // velikost work-group
   int blocks = (N + WGsize - 1) / WGsize; // počet work-group
   VecAdd<<<blocks, WGsize>>>(dA, dB, dC, N); // výpočet na GPU
   cudaMemcpy(hC, dC, size, cudaMemcpyDeviceToHost); // výsledek
   return 0;
}
```

OpenCL

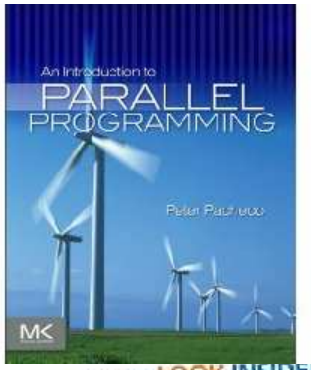
- Vyzkoušejte si demo na svém počítači
  - <stažení a překlad dema>

Shrnutí

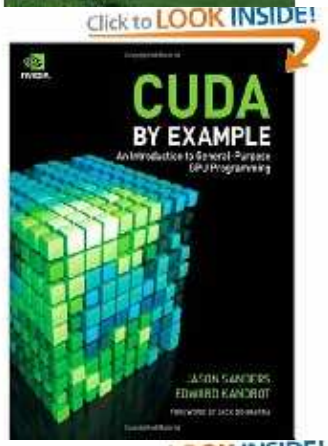
- Ukázali jsme si jak spouštět programy paralelně
- Ukázali jsme si jak optimalizovat programy pomocí knihoven na vysoké úrovni
- Ukázali jsme si základy paralelního programování na nízké úrovni

Literatura

- ***An Introduction to Parallel Programming*** od Peter Pacheco



- ***CUDA by Example: An Introduction to General-Purpose GPU Programming*** od Jason Sanders, Edward Kandrot



- ***Introduction to Parallel Computing (2nd Edition)*** od Ananth Grama, George Karypis, Vipin Kumar a Anshul Gupta