Qminers®

Digging Deep for Performance

Hloubkové dolování výkonu

Mgr. Petr Filipský

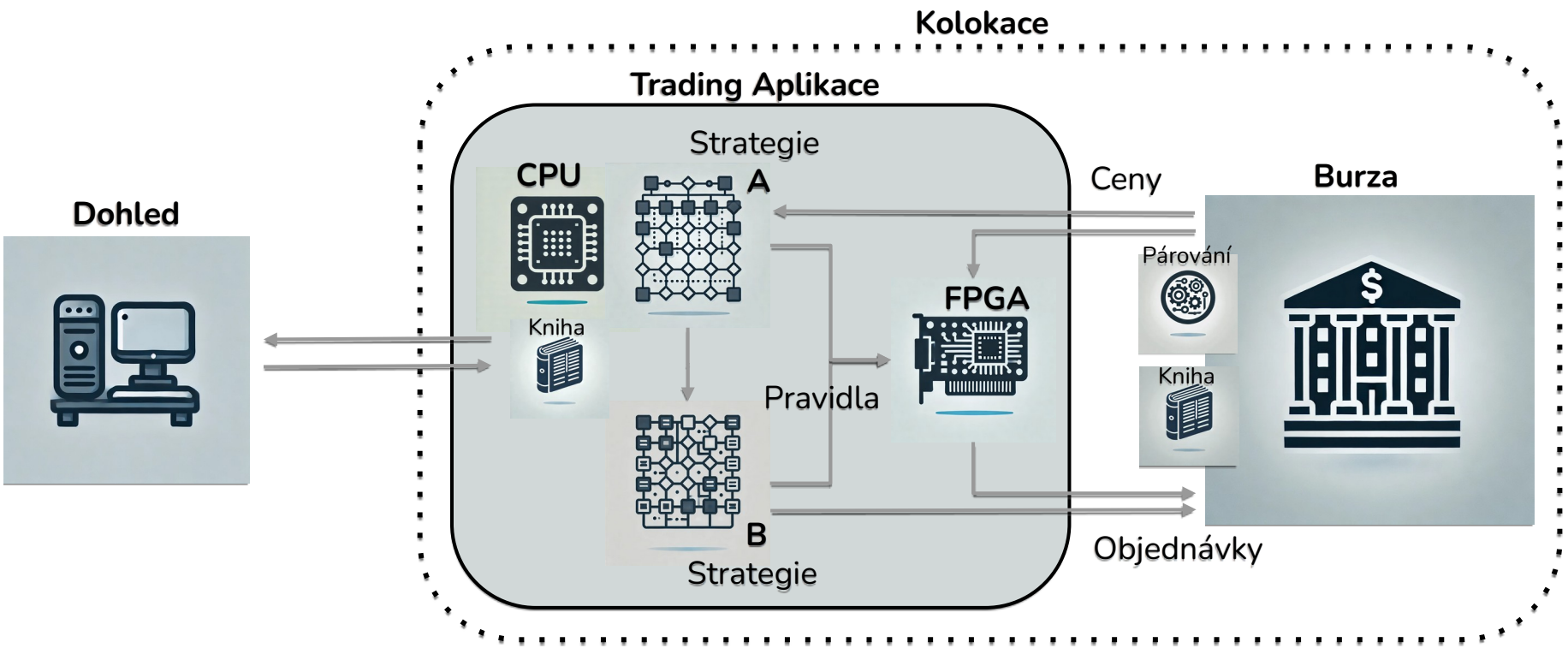
Agenda


DIGGING DEEP
FOR PERFORMANCE
Agendae of the Agenda
AGENDA
AGENDA

Agenda

- Qminers
  - Kdo jsme
  - Co vyvíjíme
  - V čem vyvíjíme
- Case studies (Pár případů z praxe)
  - Trocha teorie
  - Způsoby měření
  - Popis konkrétních situací
- Shrnutí

Motivation


Kolokace
Trading Aplikace
Strategie
CPU
A
Dohled
Ceny
Burza
Párování
FPGA
Kniha
Pravidla
Kniha
B
Objednávky
Strategie

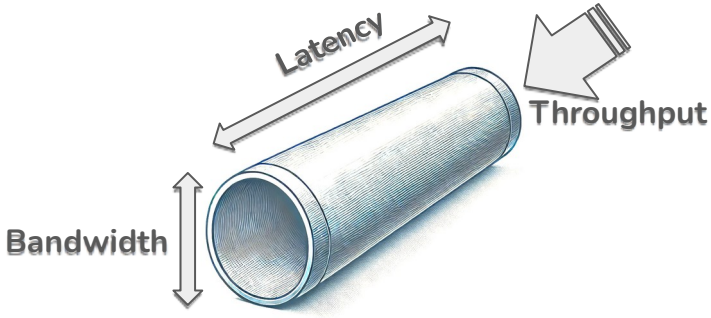
Latency / Throughput / Bandwidth

Latency / Throughput / Bandwidth

- Minimalizace **Latence**
  - Extrémně důležitá v produkci (při samotném obchodování)
  - Reakce v řádu desítek mikrosekund
    (existují HW řešení v řádu stovek případně desítek ns)

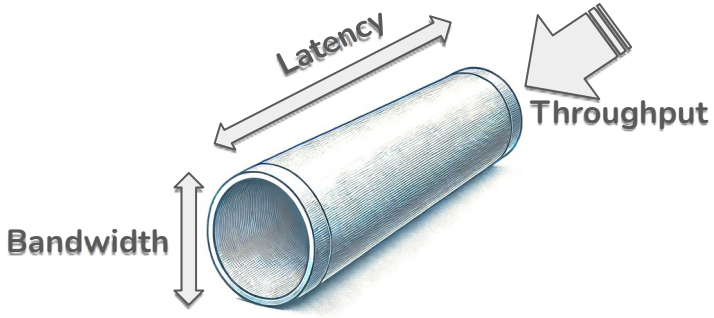
Latency / Throughput / Bandwidth

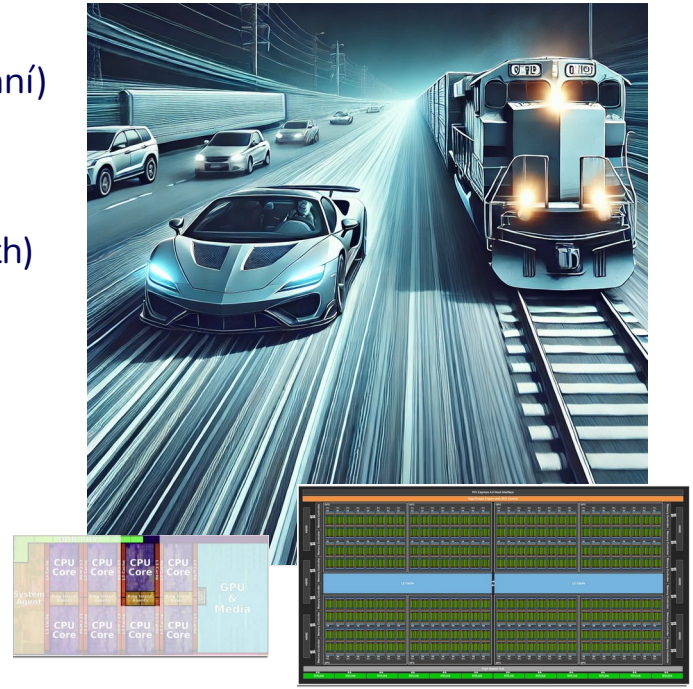
- Minimalizace **Latence**
  - Extrémně důležitá v produkci (při samotném obchodování)
  - Reakce v řádu desítek mikrosekund
    (existují HW řešení v řádu stovek případně desítek ns)
- Maximalizace **Propustnosti** (Throughput & Bandwidth)
  - Duležitá např. při roční simulaci
  - Získání výsledků v řádu desítek minut



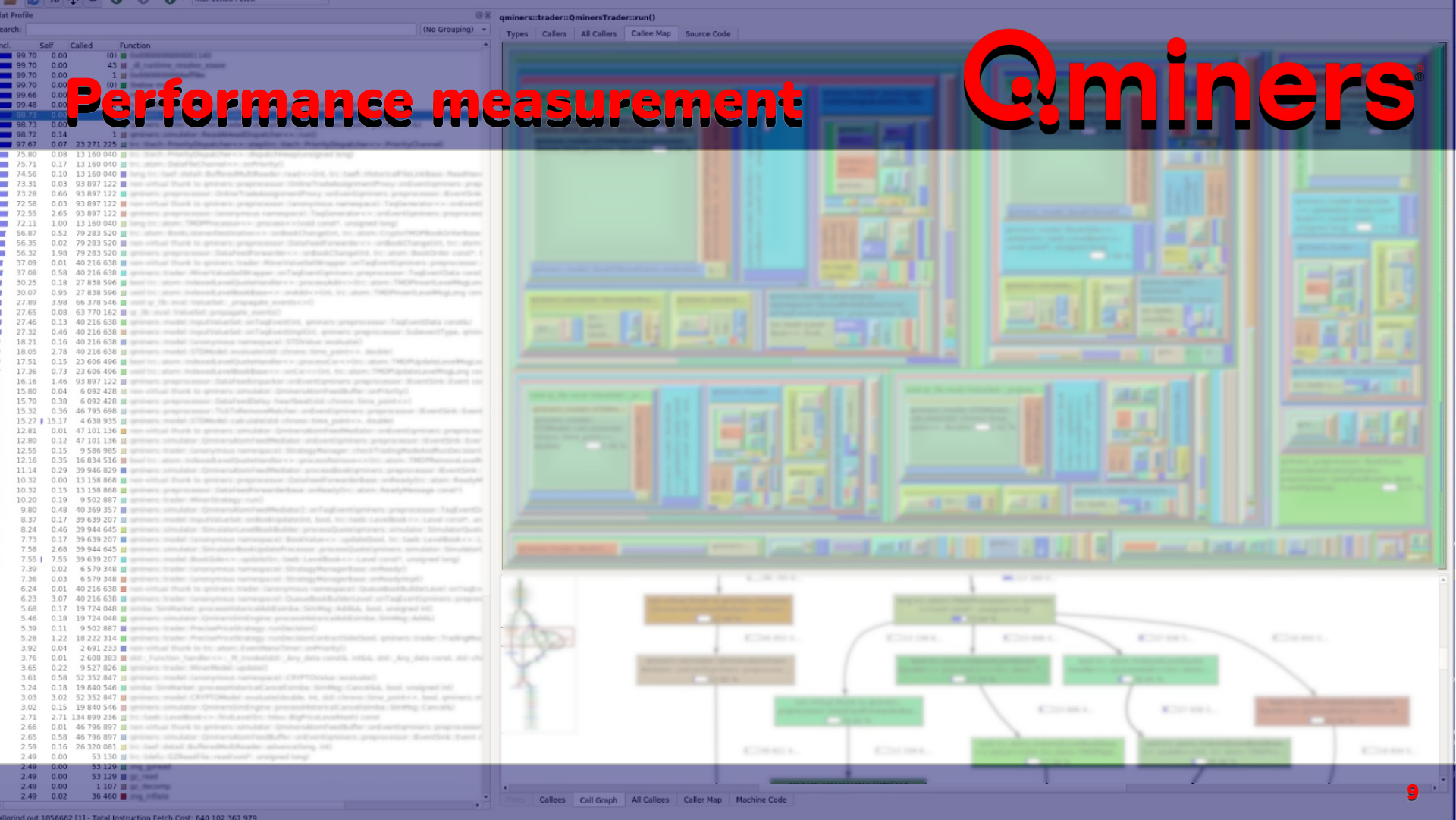
Latency / Throughput / Bandwidth

- Minimalizace **Latence**
  - Extrémně důležitá v produkci (při samotném obchodování)
  - Reakce v řádu desítek mikrosekund
    (existují HW řešení v řádu stovek případně desítek ns)
- Maximalizace **Propustnosti** (Throughput & Bandwidth)
  - Duležitá např. při roční simulaci
  - Získání výsledků v řádu desítek minut





Performance measurement



Performance measurement

Samplování (Linux perf, Intel VTune, …)

Performance measurement

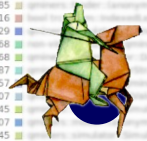
Samplování (Linux perf, Intel VTune, ...)

Instrumentace (Optick, ...)

Performance measurement

Samplování (Linux perf, Intel VTune, ...)

Instrumentace (Optick, ...)



Emulace (Valgrind)

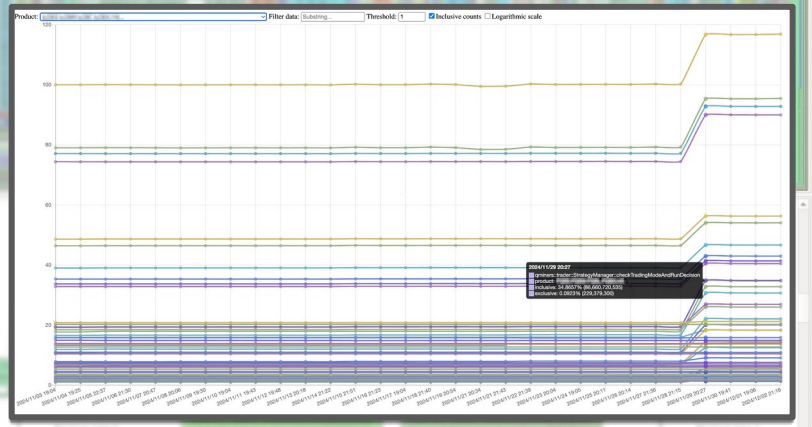
Performance measurement

Qminers

Samplování (Linux perf, Intel VTune, ...)

Instrumentace (Optick, ...)

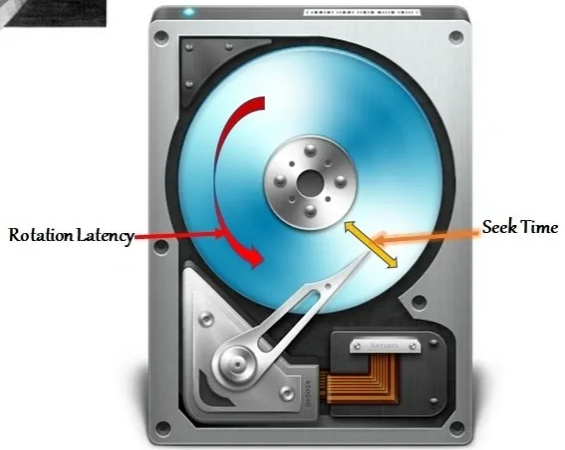
Emulace (Valgrind)



Memory

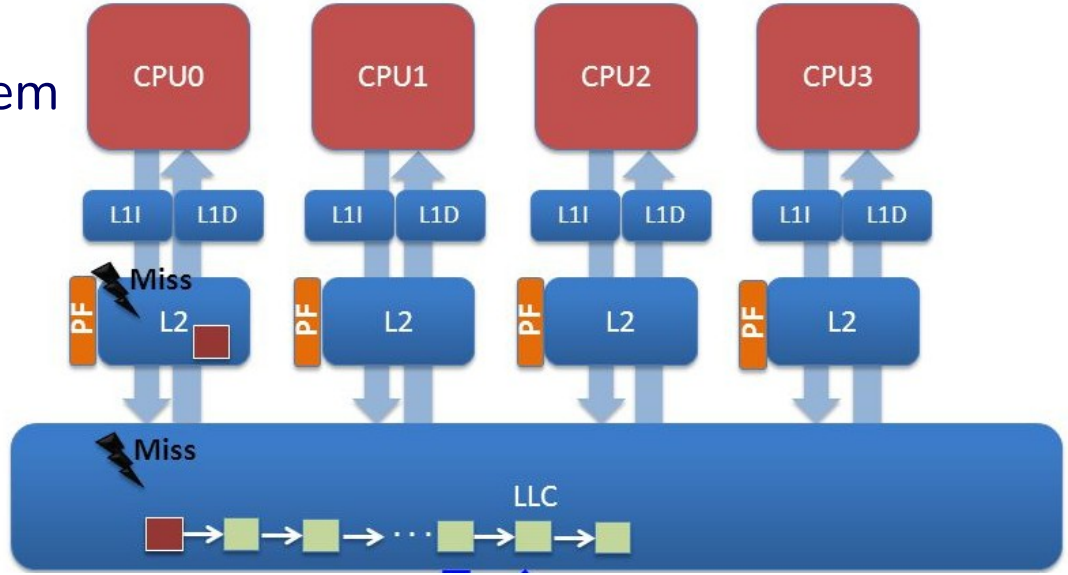
Memory

- Paměti s lineárním přístupem
  - Magnetické pásky
  - Plotnové disky



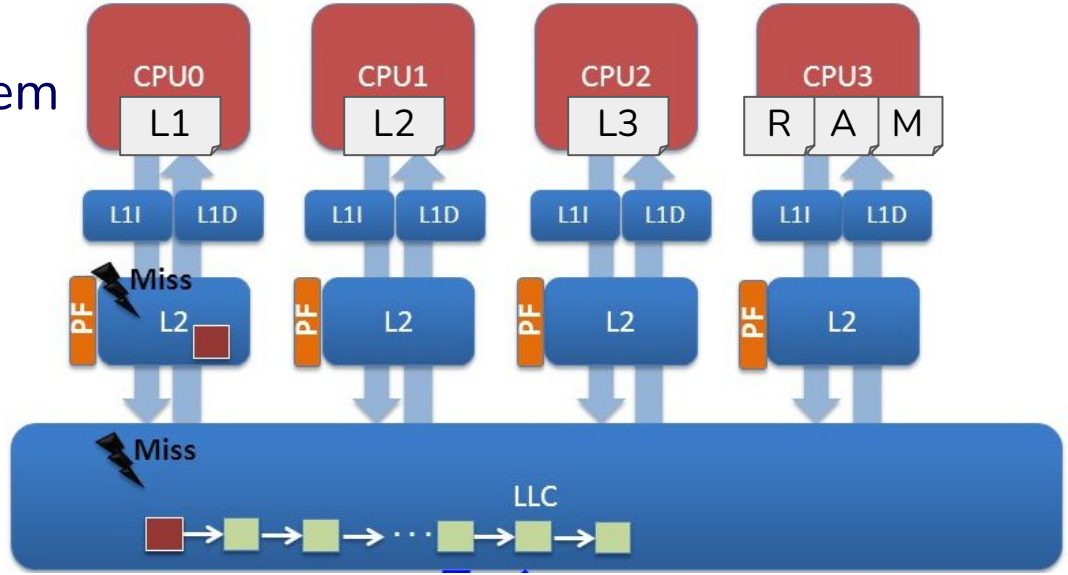
Memory

- Paměti s lineárním přístupem
  - Magnetické pásky
  - Plotnové disky
- Je RAM opravdu RAM?
  - Hierarchie vyrovnávacích pamětí
  - Různé latence



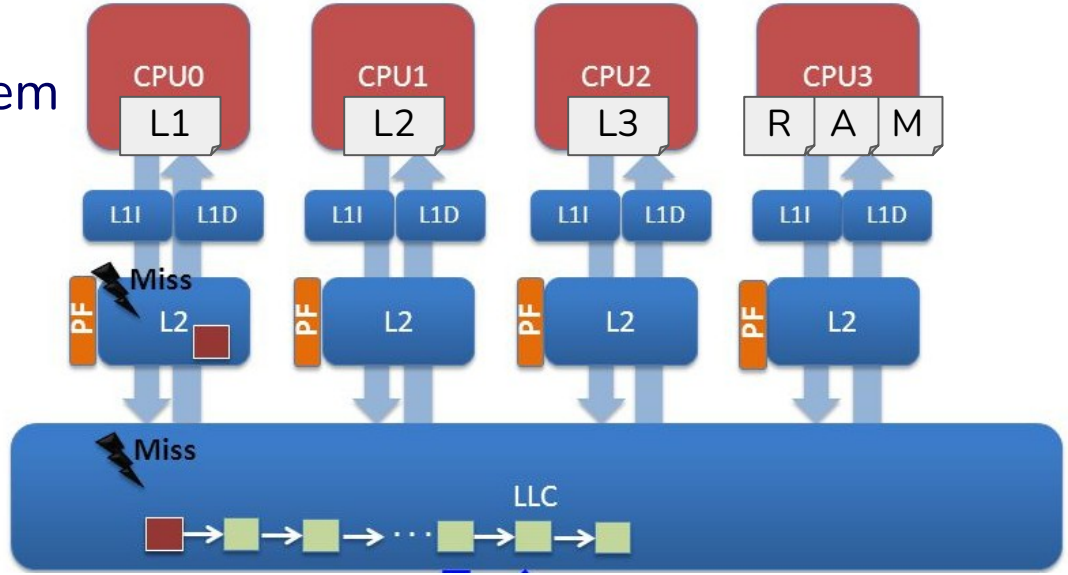
Memory

- Paměti s lineárním přístupem
  - Magnetické pásky
  - Plotnové disky
- Je RAM opravdu RAM?
  - Hierarchie vyrovnávacích pamětí
  - Různé latence



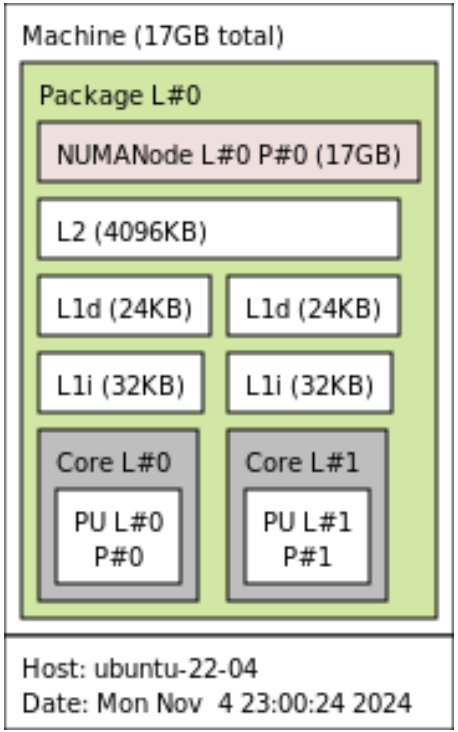
Memory

- Paměti s lineárním přístupem
  - Magnetické pásky
  - Plotnové disky
- Je RAM opravdu RAM?
  - Hierarchie vyrovnávacích pamětí
  - Různé latence
- Lokalita přístupu
  - **Časová** (Cache)
  - **Prostorová** (Cache line a Prefetcher)

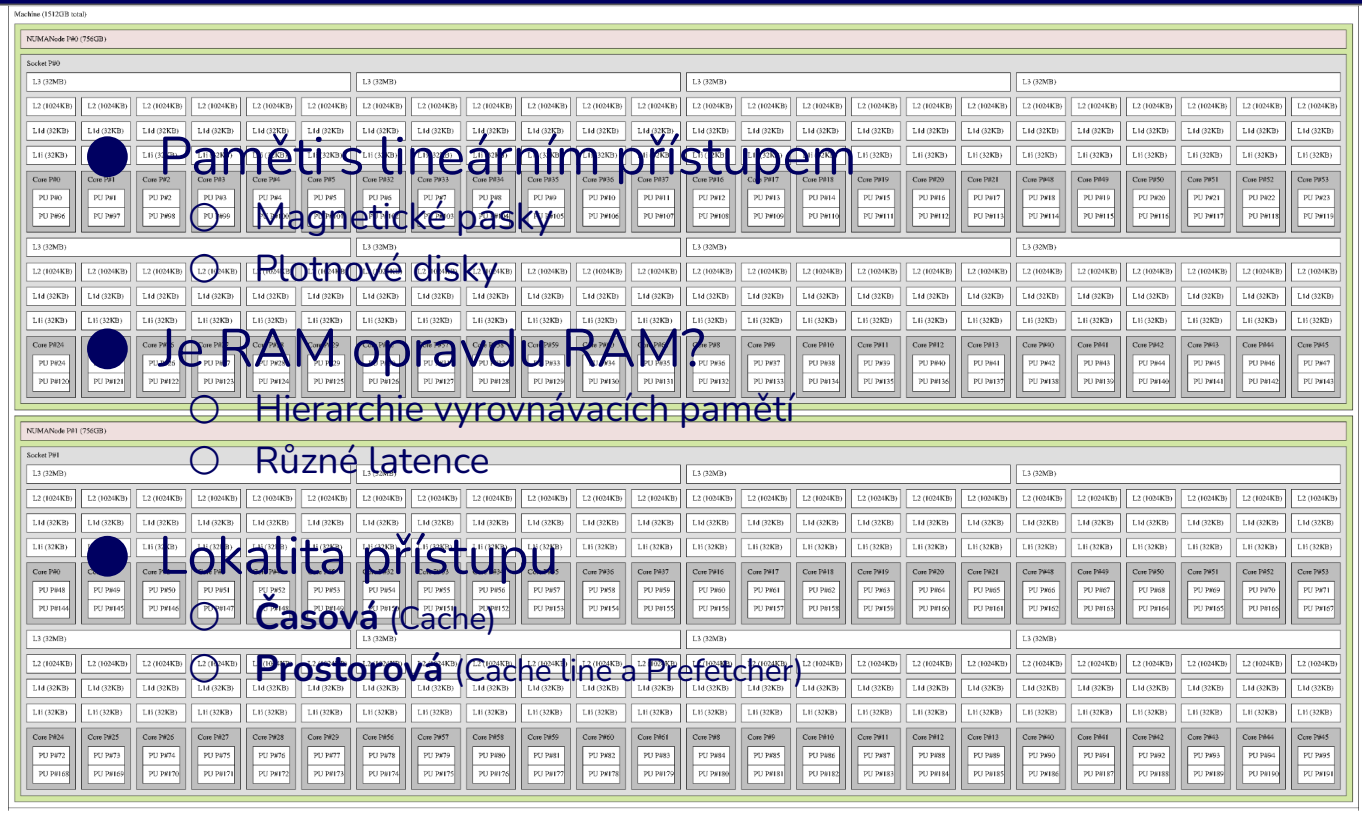


Memory

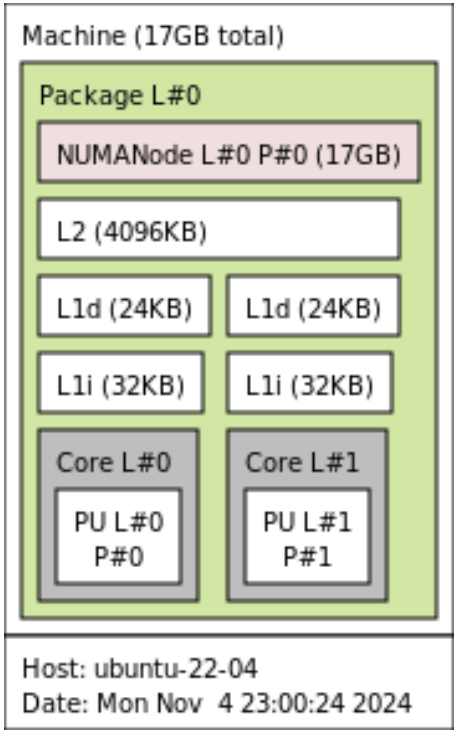
- Paměti s lineárním přístupem
  - Magnetické pásky
  - Plotnové disky
- Je RAM opravdu RAM?
  - Hierarchie vyrovnávacích pamětí
  - Různé latence
- Lokalita přístupu
  - **Časová** (Cache)
  - **Prostorová** (Cache line a Prefetcher)



Memory

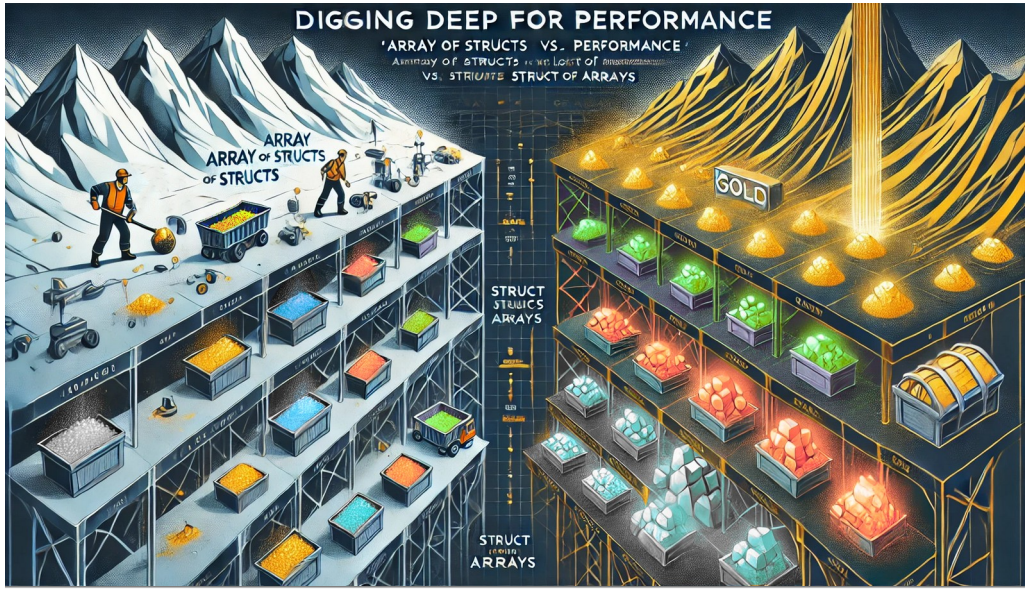


- Paměti s lineárním přístupem
  - Magnetické pásky
  - Plotnové disky
- Je RAM opravdu RAM?
  - Hierarchie vyrovnávacích pamětí
  - Různé latence
- Lokalita přístupu
  - **Časová** (Cache)
  - **Prostorová** (Cache-line a Prefetcher)



Data Oriented Design

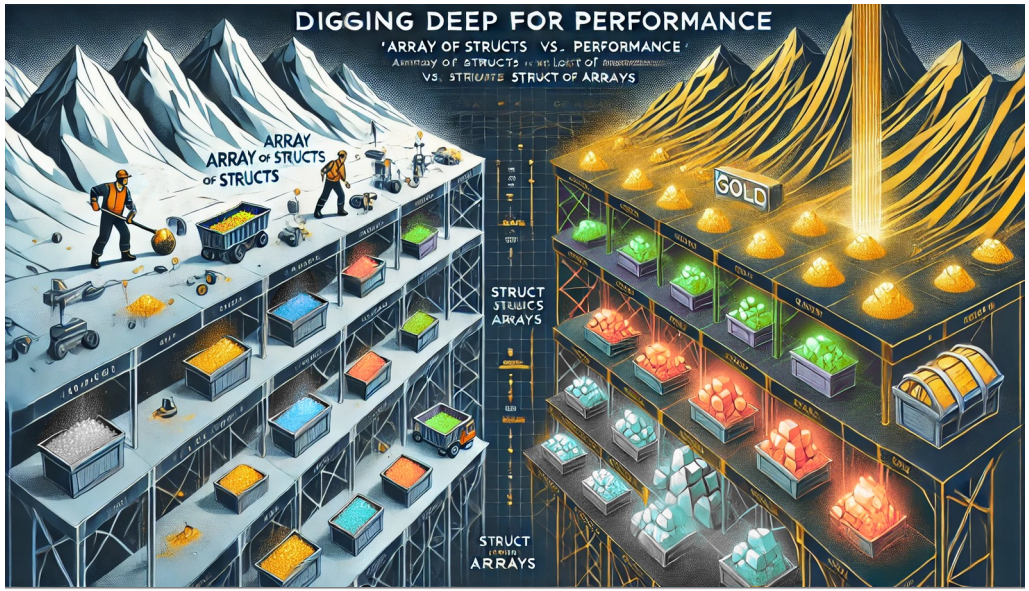
Qminers®



Data Oriented Design

- Array of Structs

```
namespace AOS {
struct Beast {
    Point3D pos{};
    Point3D vel{};
    HealthPoints hp{};
};
using Beasts = std::vector<Beast>;
```



Credit: *Mathieu Ropert - Data Oriented Design and ECS Explained*

Data Oriented Design

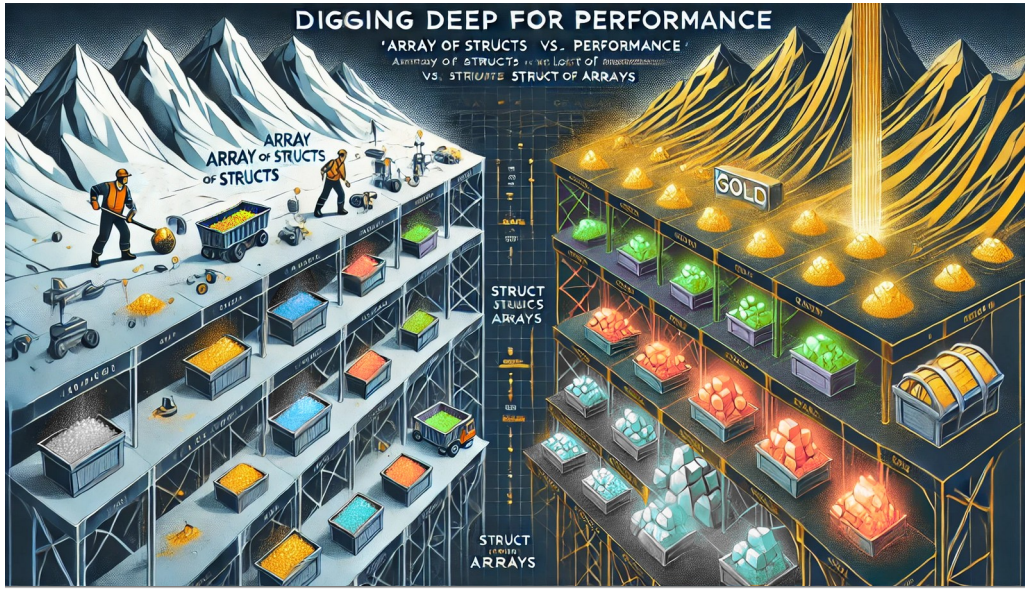
Qminers

- Array of Structs

```cpp
namespace AOS {
struct Beast {
    Point3D pos{};
    Point3D vel{};
    HealthPoints hp{};
};
using Beasts = std::vector<Beast>;
```

- Struct of Arrays

```cpp
namespace SOA {
struct Beasts {
    explicit Beasts(size_t size)
      : pos(size), vel(size), hp(size) {
    }
    std::vector<Point3D> pos;
    std::vector<Point3D> vel;
    std::vector<HealthPoints> hp;
};
```



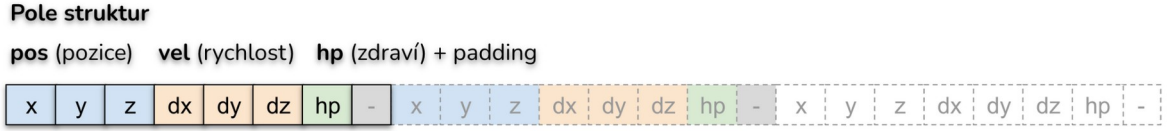
Credit: *Mathieu Ropert - Data Oriented Design and ECS Explained*

Data Oriented Design

Qminers

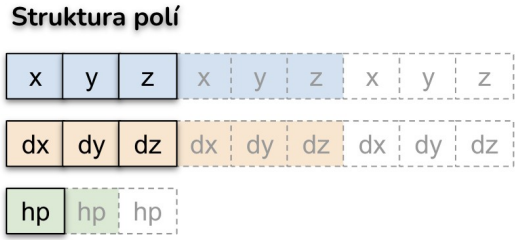
- Array of Structs

```
namespace AOS {
struct Beast {
    Point3D pos{};
    Point3D vel{};
    HealthPoints hp{};
};
using Beasts = std::vector<Beast>;
```



- Struct of Arrays

```
namespace SOA {
struct Beasts {
    explicit Beasts(size_t size)
      : pos(size), vel(size), hp(size) {
    }
    std::vector<Point3D> pos;
    std::vector<Point3D> vel;
    std::vector<HealthPoints> hp;
};
```



Credit: *Mathieu Ropert - Data Oriented Design and ECS Explained*

Data Oriented Design

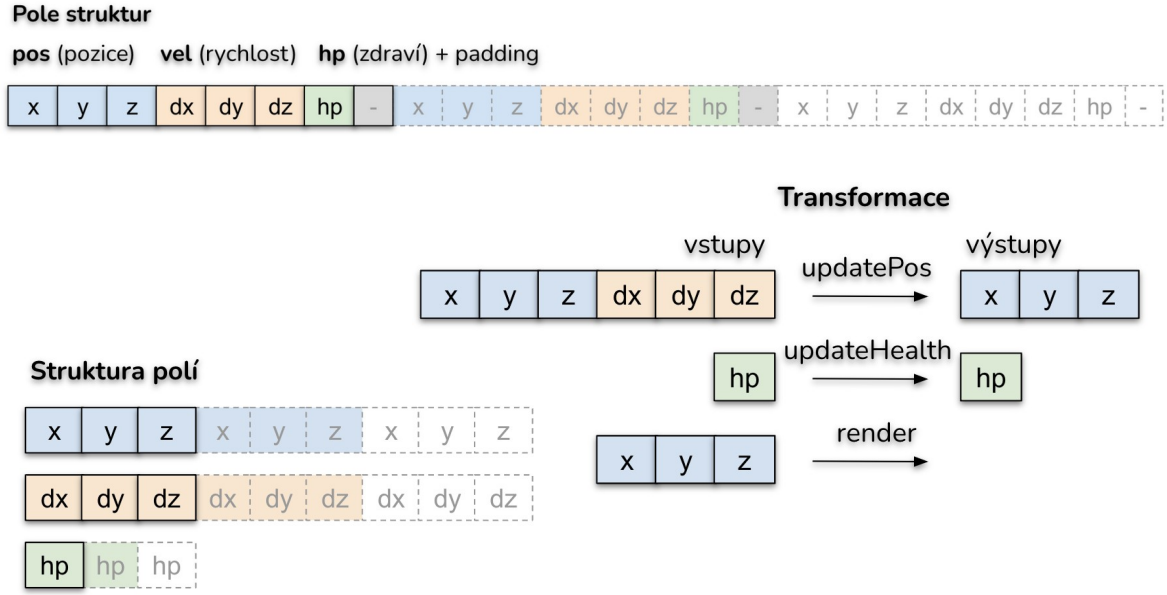
Qminers

- Array of Structs

```
namespace AOS {
struct Beast {
    Point3D pos{};
    Point3D vel{};
    HealthPoints hp{};
};
using Beasts = std::vector<Beast>;
```

- Struct of Arrays

```
namespace SOA {
struct Beasts {
    explicit Beasts(size_t size)
      : pos(size), vel(size), hp(size) {
    }
    std::vector<Point3D> pos;
    std::vector<Point3D> vel;
    std::vector<HealthPoints> hp;
};
```



Godbolt example (QuickBench)

Credit: *Mathieu Ropert - Data Oriented Design and ECS Explained*

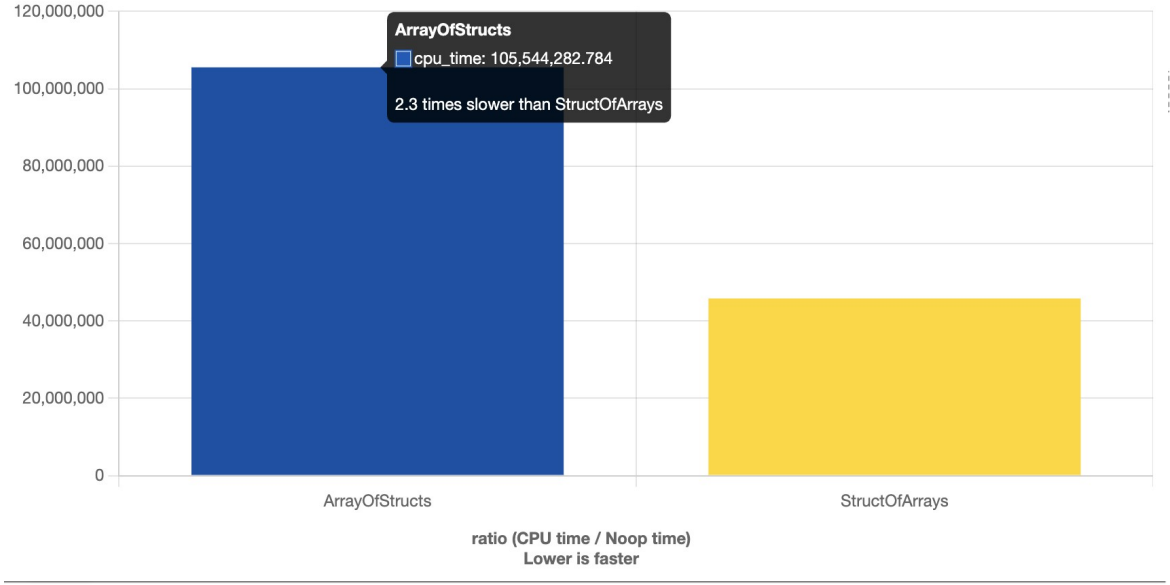
Data Oriented Design

- Array of Structs

```
namespace AOS {
struct Beast {
    Point3D pos{};
    Point3D vel{};
    HealthPoints hp{};
};
using Beasts = std::vector<Beast>;
```

- Struct of Arrays

```
namespace SOA {
struct Beasts {
    explicit Beasts(size_t size)
      : pos(size), vel(size), hp(size) {
    }
    std::vector<Point3D> pos;
    std::vector<Point3D> vel;
    std::vector<HealthPoints> hp;
};
```

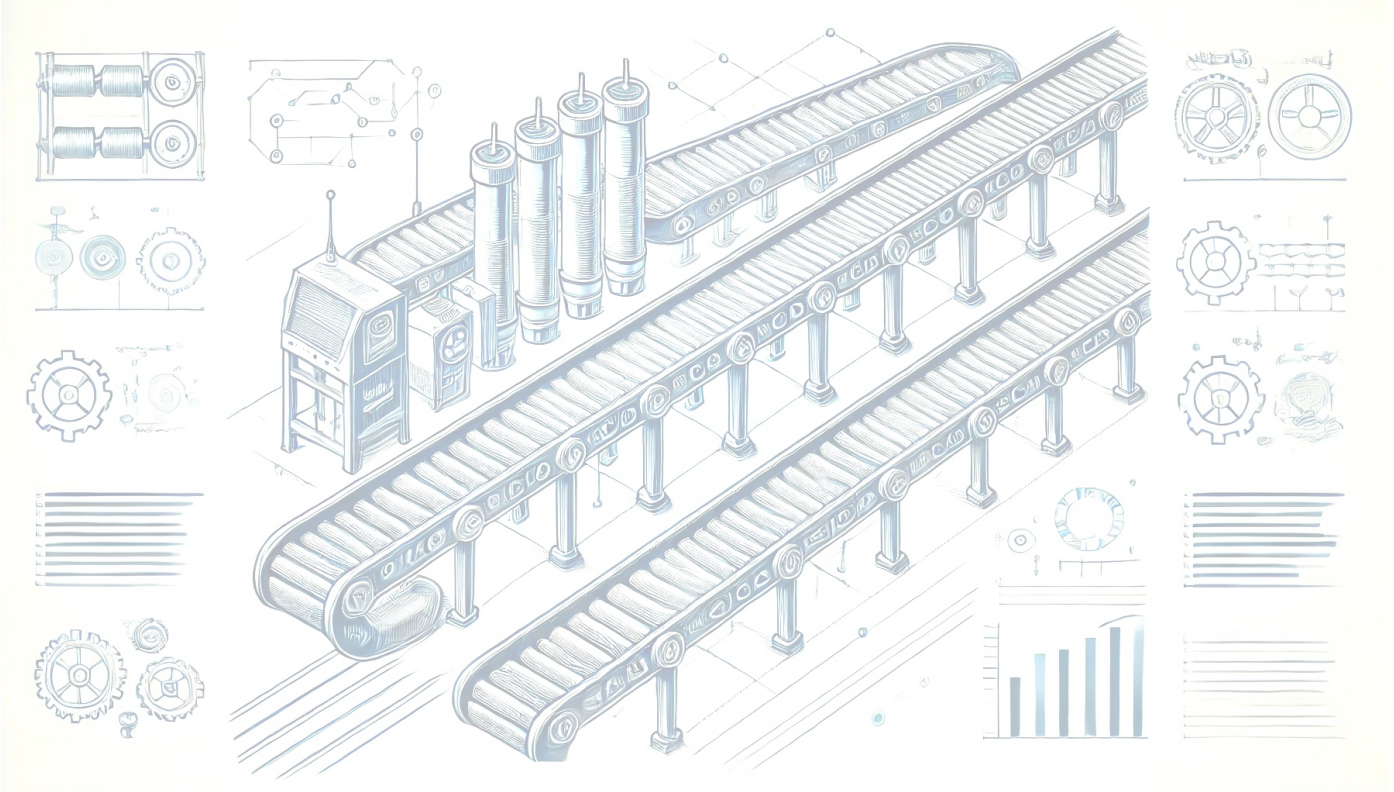


Godbolt example (QuickBench)

Credit: *Mathieu Ropert - Data Oriented Design and ECS Explained*

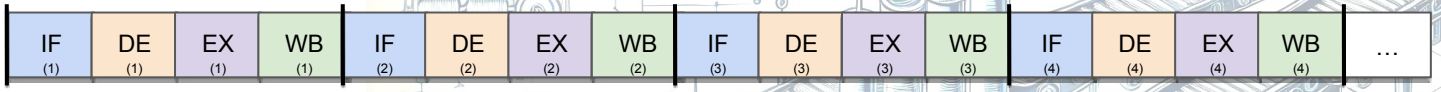
CPU Pipeline

Qminers®



CPU Pipeline

- Sekvenční zpracování

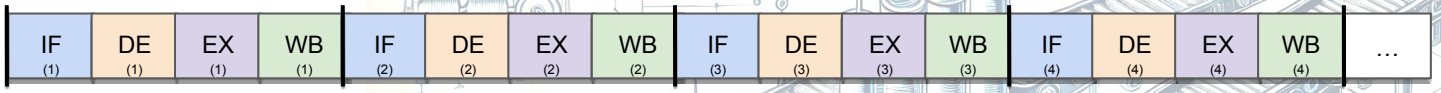


Credit: *Jobin Johnson - Branchless programming. Does it really matter?*

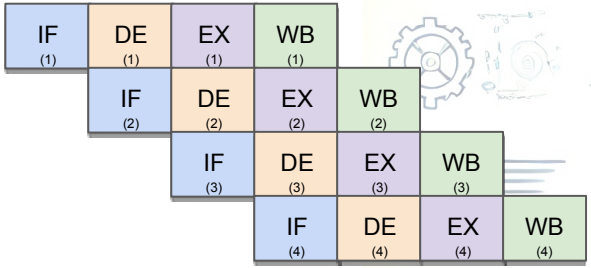
CPU Pipeline

Qminers

- Sekvenční zpracování



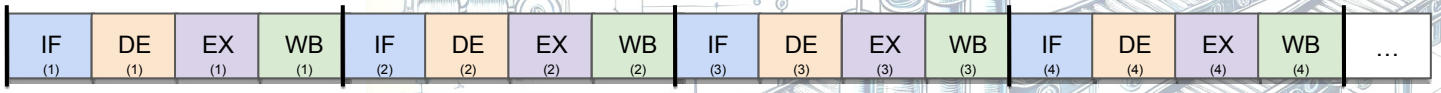
- Pipeline zpracování



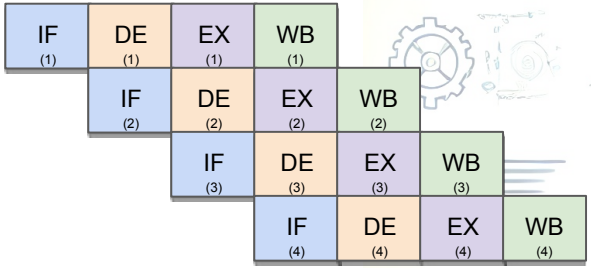
Credit: *Jobin Johnson - Branchless programming. Does it really matter?*

CPU Pipeline

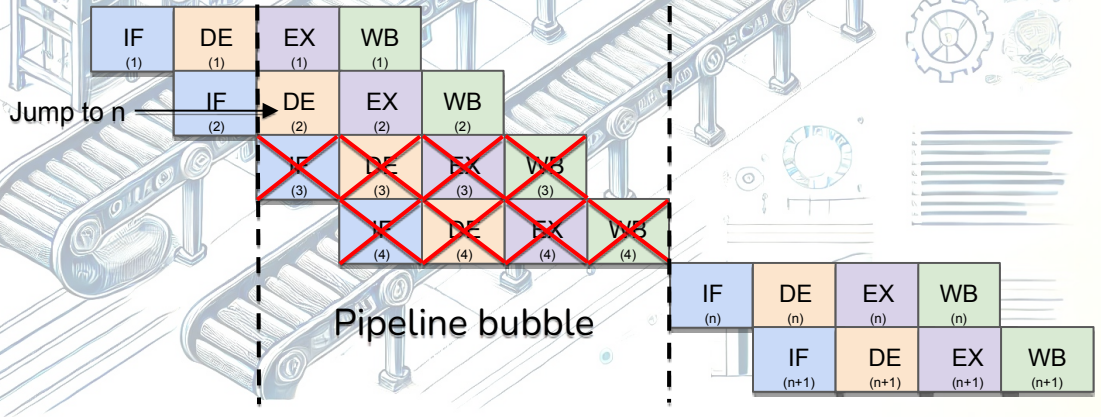
- **Sekvenční zpracování**



- **Pipeline zpracování**



- **Podmíněný skok** (nesprávně predikovaný)



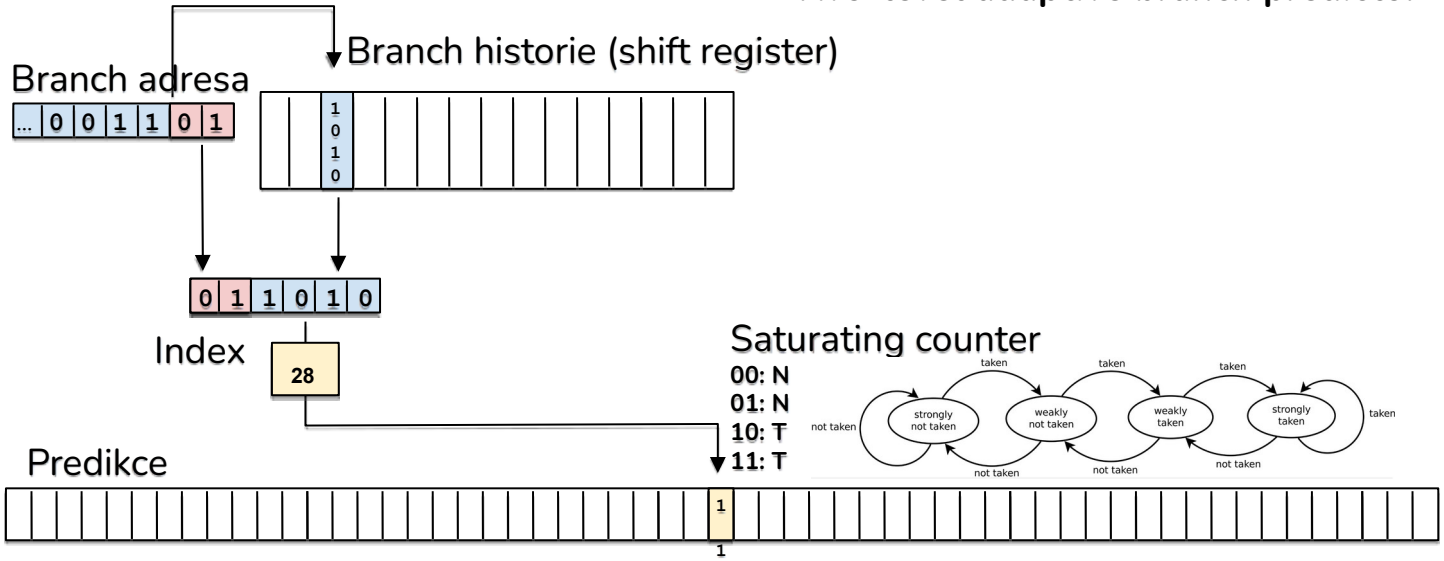
Credit: *Jobin Johnson - Branchless programming. Does it really matter?*

Branch Predictor

Branch Predictor

- Branch Predictor
  - Obvod v CPU, který předpovídá výsledky podmínek v kódu

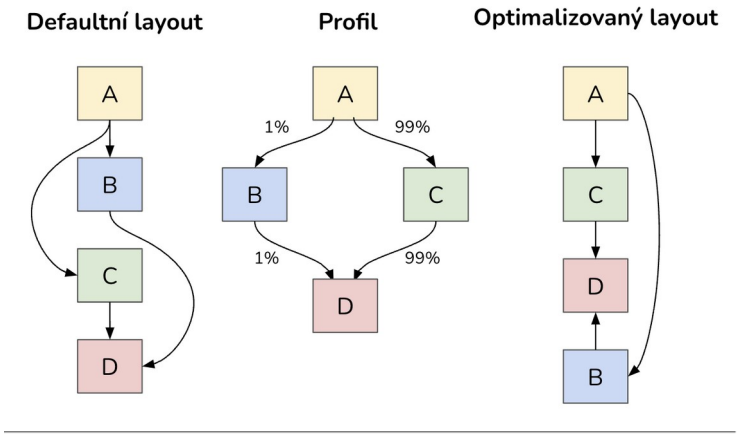
Two-level adaptive branch predictor



Credit: *Dan Luu - Branch prediction*

Branch Predictor

- Branch Predictor
  - Obvod v CPU, který předpovídá výsledky podmínek v kódu
- Profile-guided optimization
  - Optimalizace kódu na základě statistik získaných při běhu programu



Branch Predictor

- Branch Predictor
  - Obvod v CPU, který předpovídá výsledky podmínek v kódu
- Profile-guided optimization
  - Optimalizace kódu na základě statistik získaných při běhu programu
- Branch Latency

```
1. def handle_trade(trade):
2.     if trade.size > threshold:
3.         send_order_triggered_by_trade(trade)
```

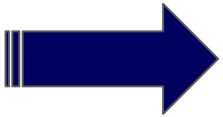
Branch Predictor

- Branch Predictor
  - Obvod v CPU, který předpovídá výsledky podmínek v kódu
- Profile-guided optimization
  - Optimalizace kódu na základě statistik získaných při běhu programu
- Branch Latency
  - CPU i překladače občas pracují “proti nám”
  - Řešením je použití specializovaného HW (FPGA, ...)

Branchless Programming

Branchless Programming

```
>>> def side_to_sign(isSell : bool) -> int:
...     return 1 if isSell else -1
...
>>> side_to_sign(True)
1
>>> side_to_sign(False)
-1
```

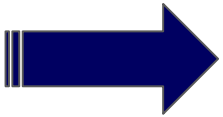


```
>>> def side_to_sign(isSell : bool) -> int:
...     return 2 * isSell - 1
...
>>> side_to_sign(True)
1
>>> side_to_sign(False)
-1
```

Branchless Programming

Qminers

```
>>> def side_to_sign(isSell : bool) -> int:
...     return 1 if isSell else -1
...
>>> side_to_sign(True)
1
>>> side_to_sign(False)
-1
```



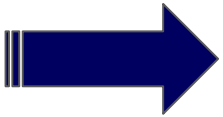
```
>>> def side_to_sign(isSell : bool) -> int:
...     return 2 * isSell - 1
...
>>> side_to_sign(True)
1
>>> side_to_sign(False)
-1
```

```
>>> def get_side(multiplier: float, isSell : bool) -> bool:
...     return not isSell if multiplier >= 0.0 else isSell
...
>>> get_side(-0.5, True)
True
>>> get_side(0.5, True)
False
>>> get_side(-0.5, False)
False
>>> get_side(0.5, False)
True
```

```
>>> def get_side(multiplier: float, isSell : bool) -> bool:
...     return (multiplier >= 0.0) ^ isSell
...
>>> get_side(-0.5, True)
True
>>> get_side(0.5, True)
False
>>> get_side(-0.5, False)
False
>>> get_side(0.5, False)
True
```

Branchless Programming

```
>>> def side_to_sign(isSell : bool) -> int:
...     return 1 if isSell else -1
...
>>> side_to_sign(True)
1
>>> side_to_sign(False)
-1
```



```
>>> def side_to_sign(isSell : bool) -> int:
...     return 2 * isSell - 1
...
>>> side_to_sign(True)
1
>>> side_to_sign(False)
-1
```

```
>>> def get_side(multiplier: float, isSell : bool) -> bool:
...     return not isSell if multiplier >= 0.0 else isSell
...
>>> get_side(-0.5, True)
True
>>> get_side(0.5, True)
False
>>> get_side(-0.5, False)
False
>>> get_side(0.5, False)
True
```

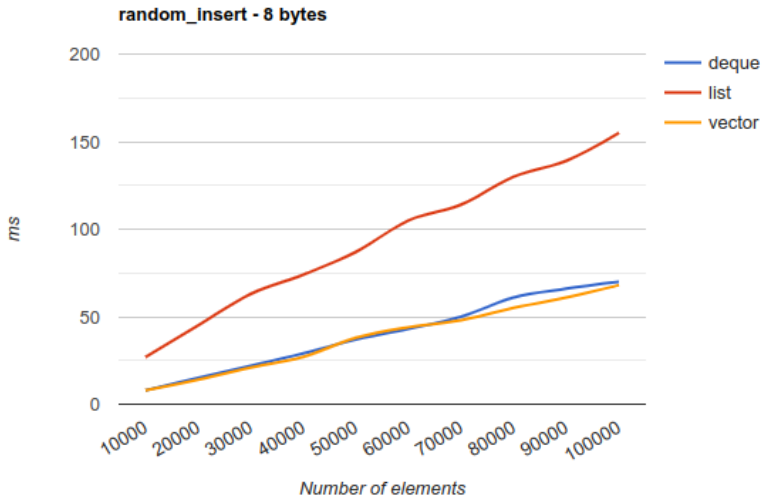
```
>>> def get_side(multiplier: float, isSell : bool) -> bool:
...     return (multiplier >= 0.0) ^ isSell
...
>>> get_side(-0.5, True)
True
>>> get_side(0.5, True)
False
>>> get_side(-0.5, False)
False
>>> get_side(0.5, False)
True
```

Godbolt example

Godbolt example (llvm-mca & flow-graph)

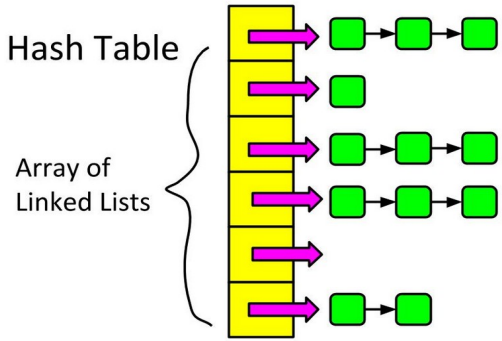
Data Structures

- Pole
  - Naprostá většina datových struktur



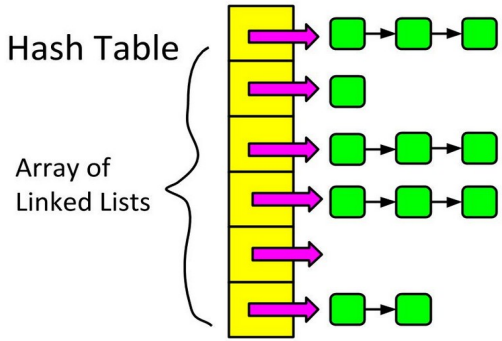
Data Structures

- Pole
  - Naprostá většina datových struktur
- Hashovací tabulky



Data Structures

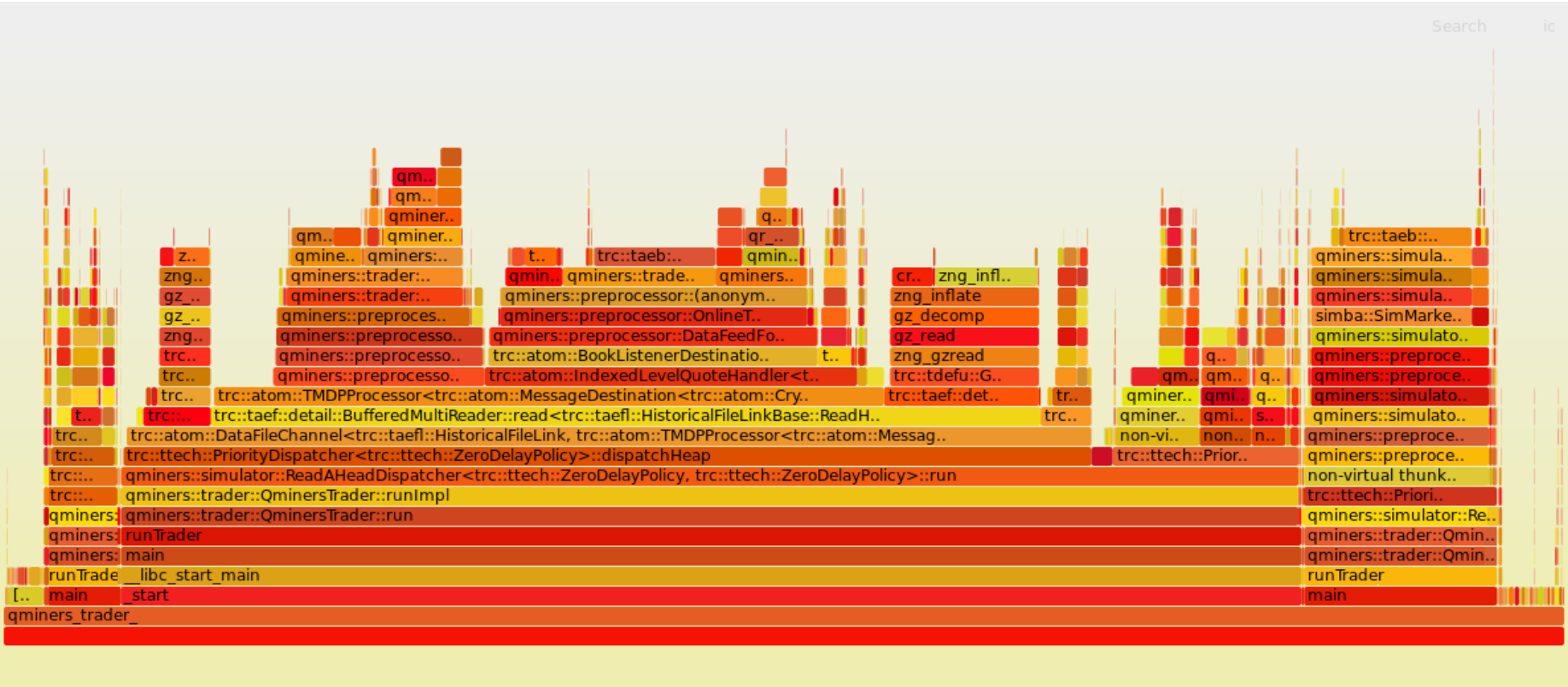
- Pole
  - Naprostá většina datových struktur
- Hashovací tabulky
  - Např. kniha limitních objednávek



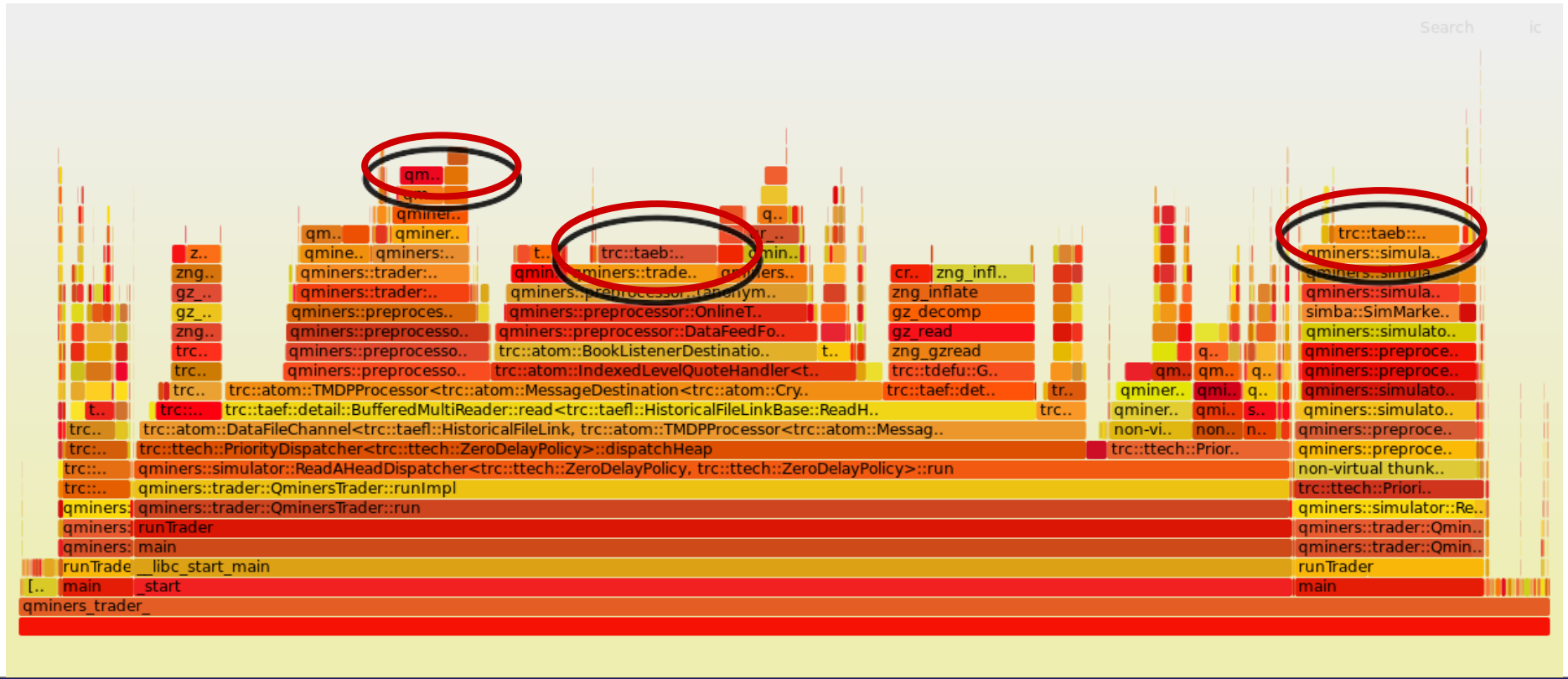
Linux perf (FlameGraph visualization)

Linux perf (FlameGraph visualization)

Qminers®

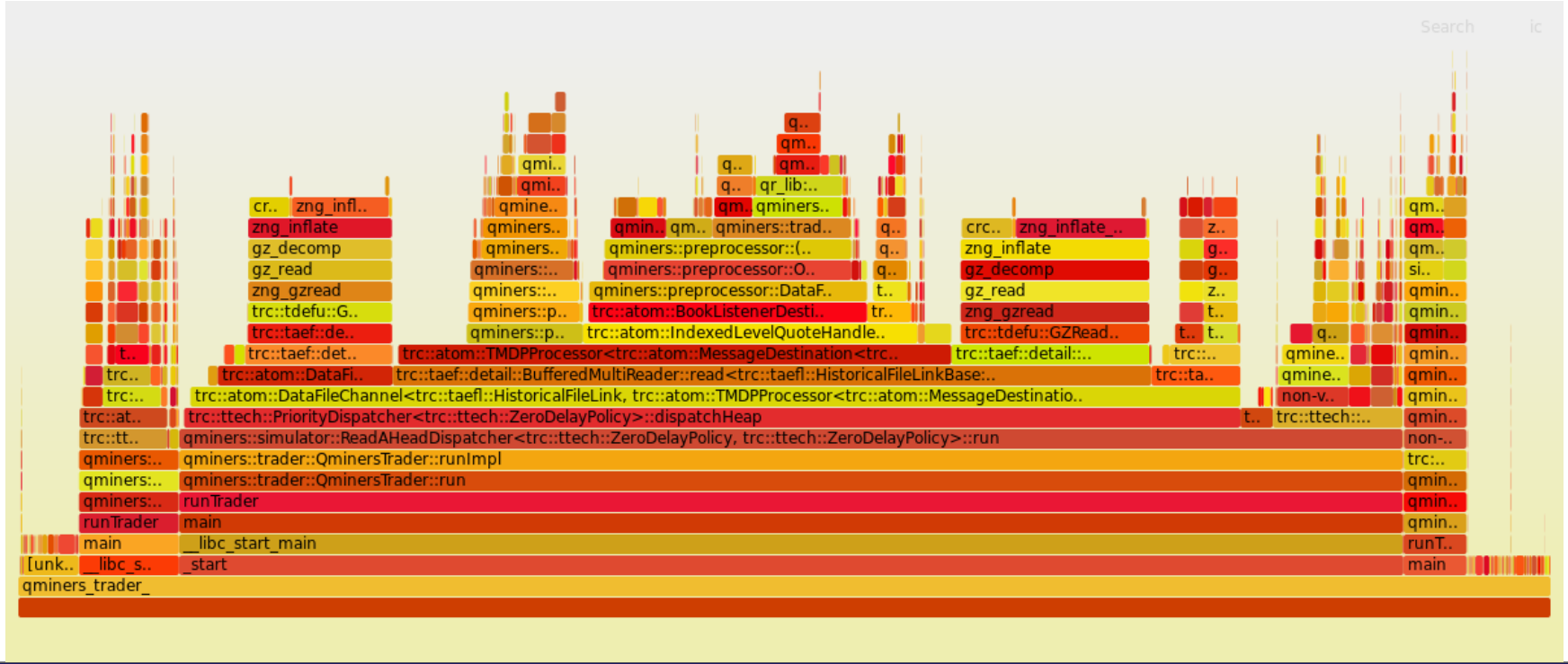


Linux perf (FlameGraph visualization)



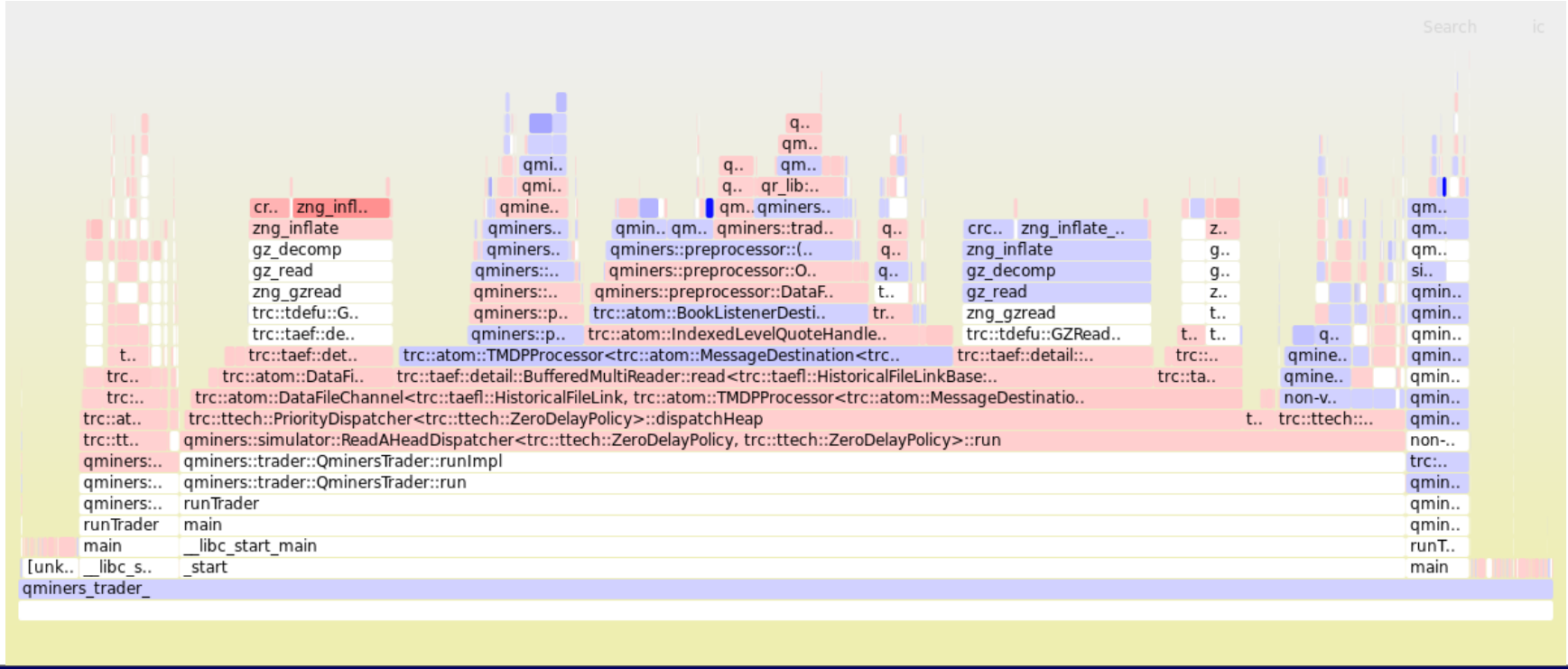
Linux perf (FlameGraph visualization)

Qminers®



Linux perf (FlameGraph visualization)

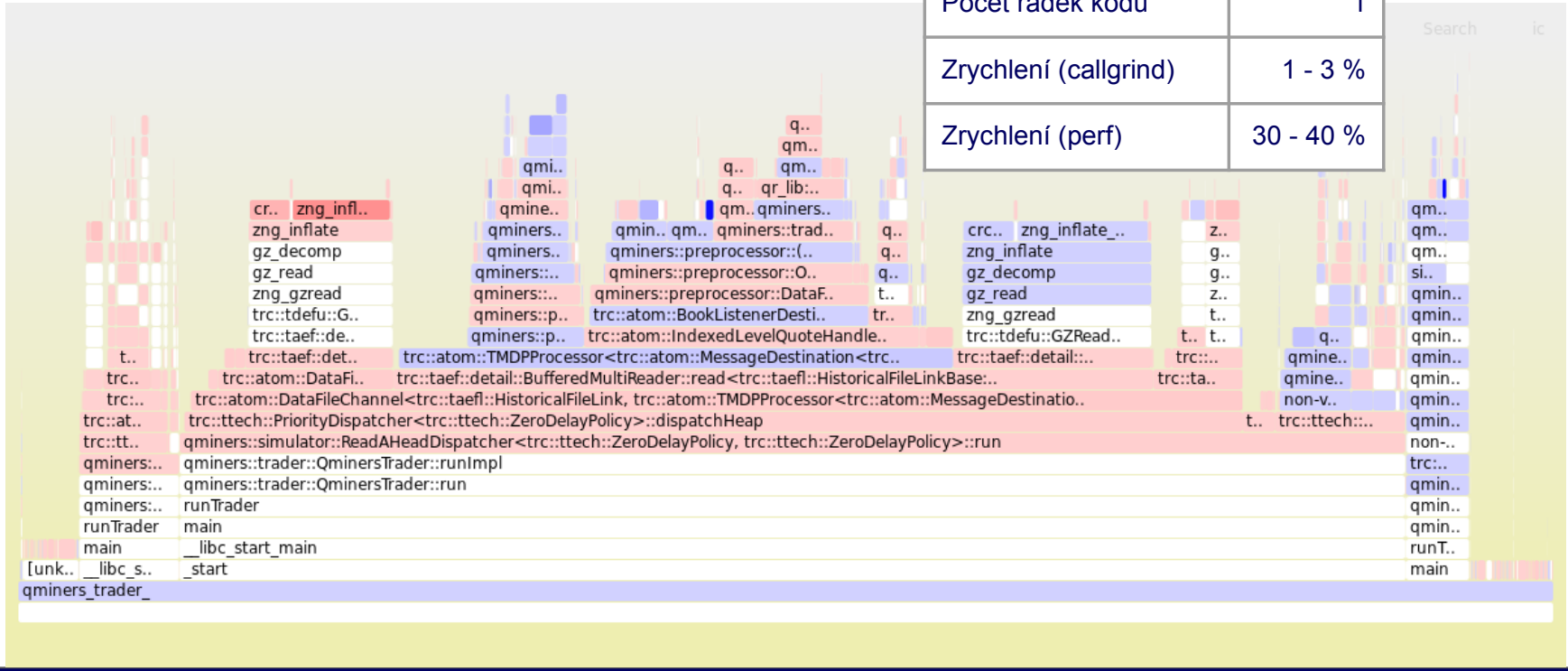
Qminers®



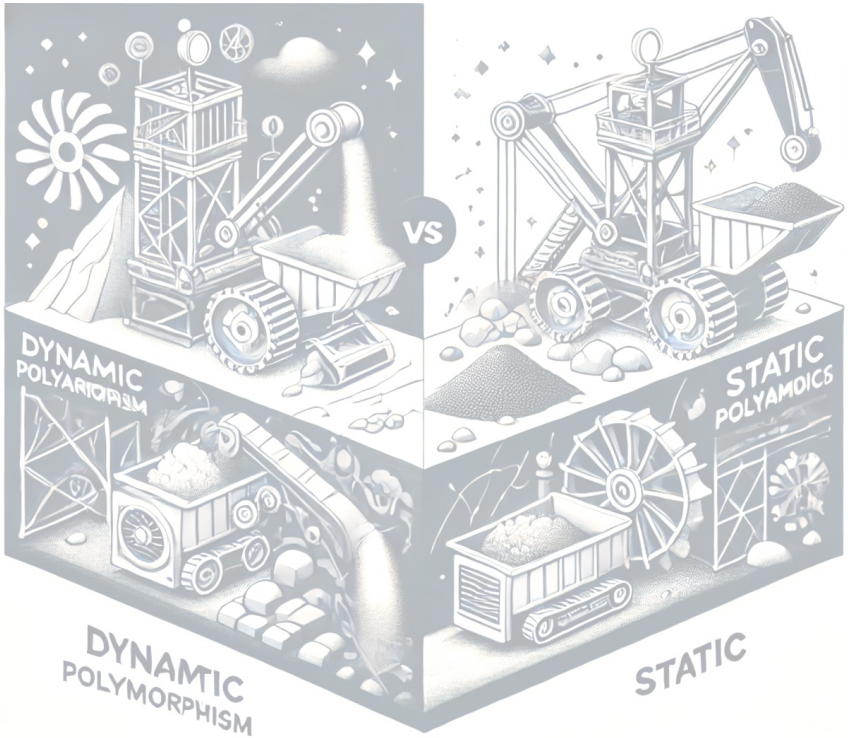
Linux perf (FlameGraph visualization)

Qminers®

Počet řádek kódu	1
Zrychlení (callgrind)	1 - 3 %
Zrychlení (perf)	30 - 40 %

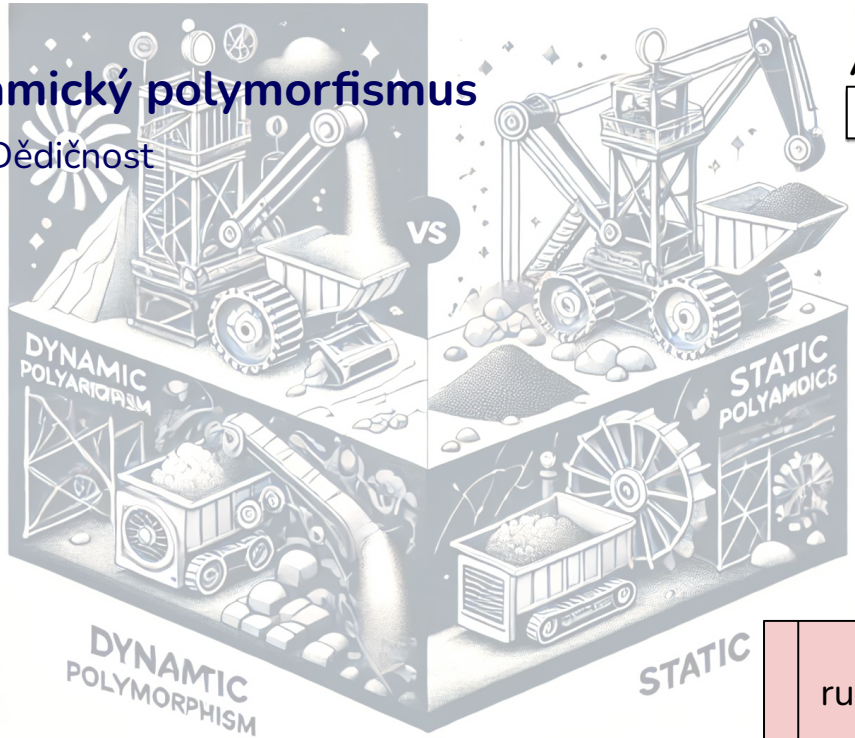


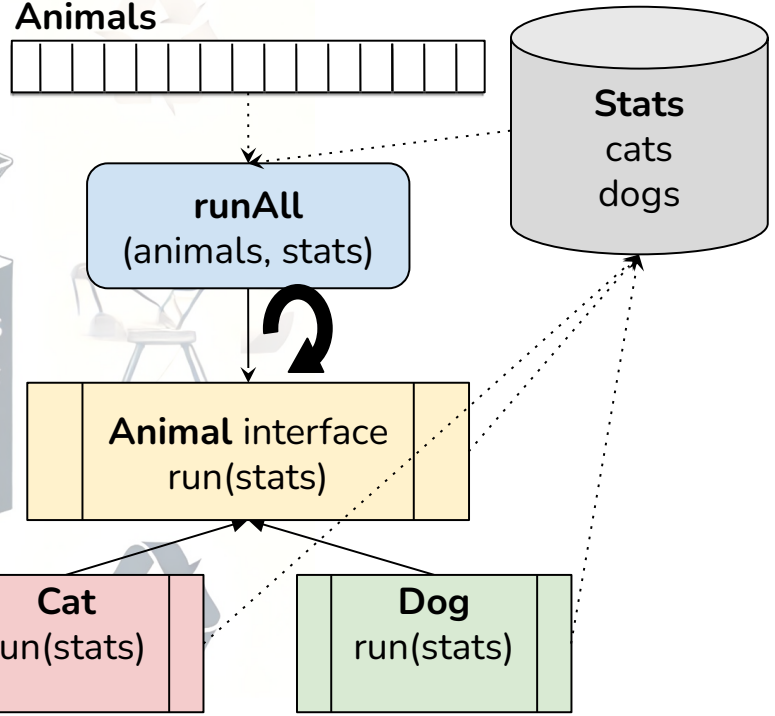
Dynamic / Static Polymorphism



Dynamic / Static Polymorphism

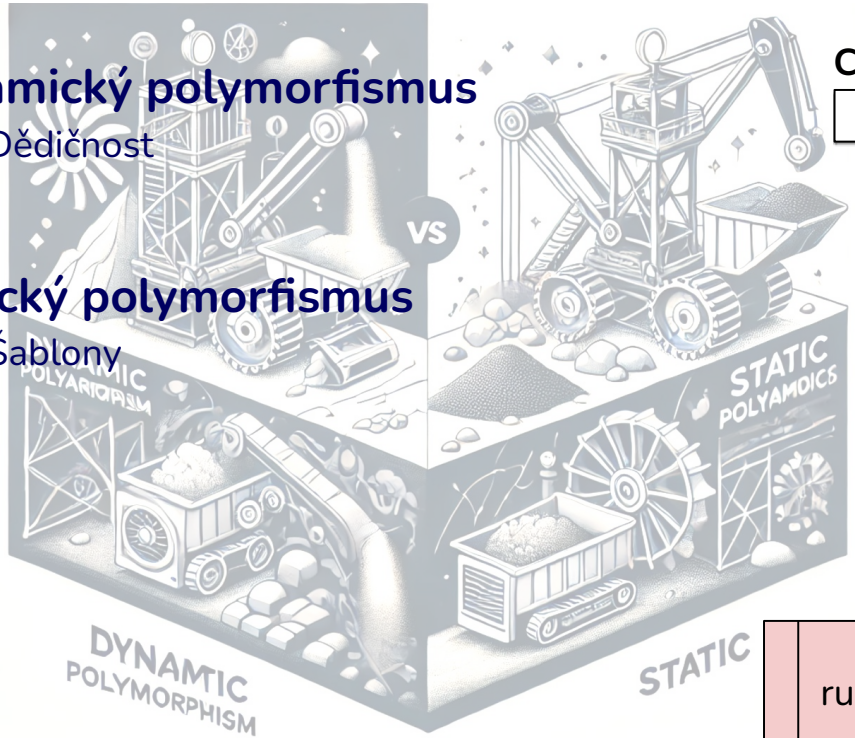
- Dynamický polymorfismus
  - Dědičnost

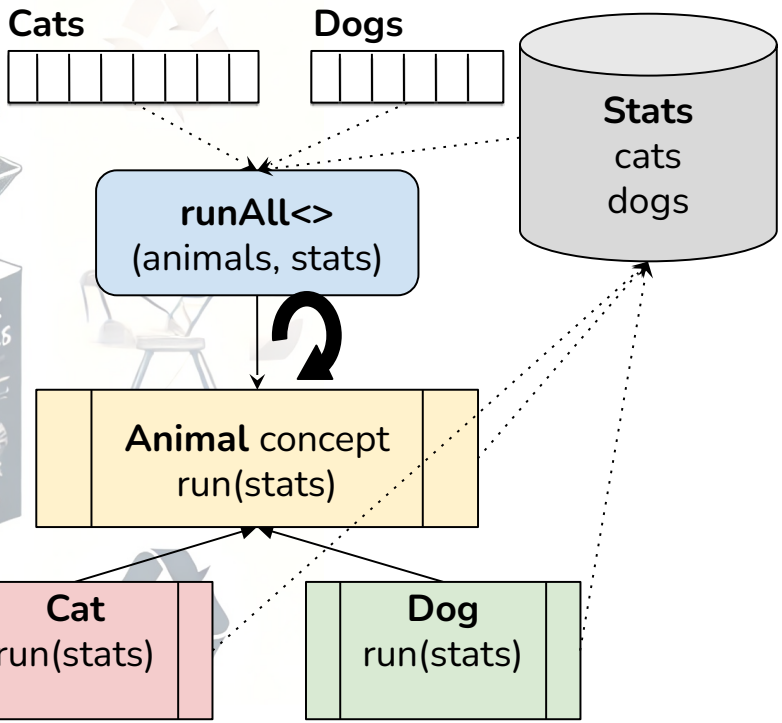




Dynamic / Static Polymorphism

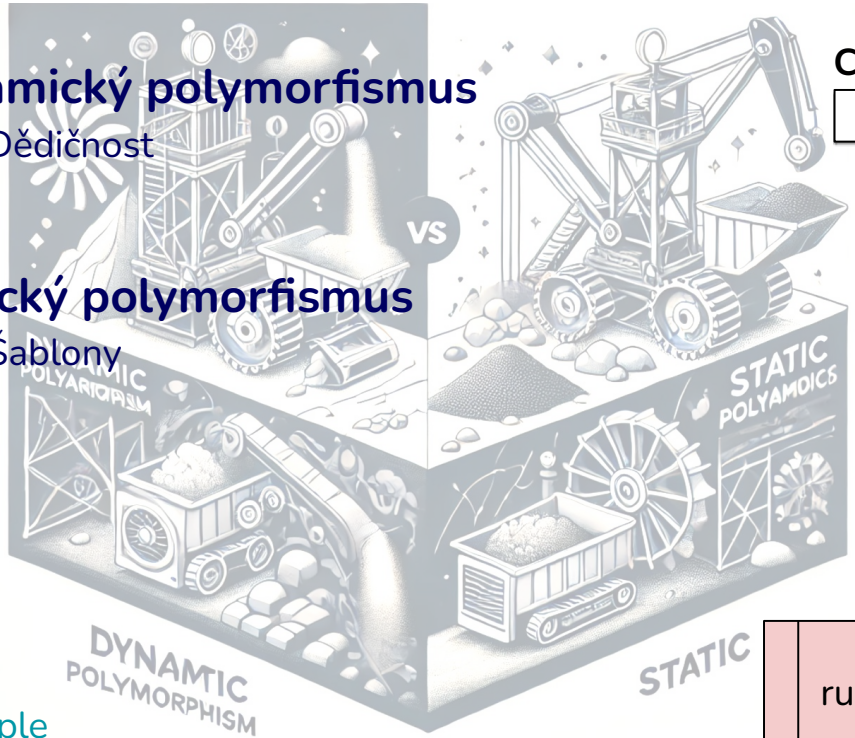
- **Dynamický polymorfismus**
  - Dědičnost
- **Statický polymorfismus**
  - Šablony

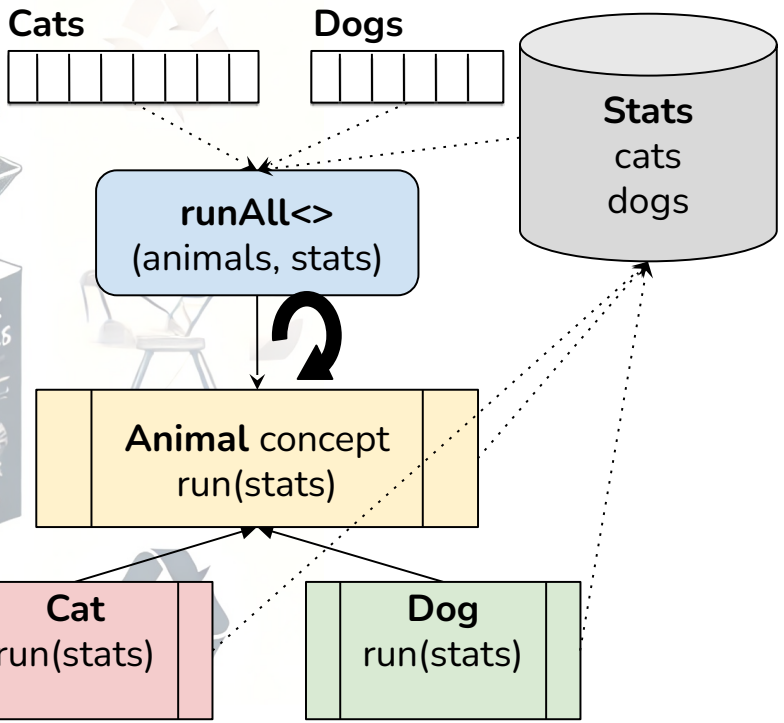




Dynamic / Static Polymorphism

- Dynamický polymorfismus
  - Dědičnost
- Statický polymorfismus
  - Šablony





[Godbolt example]

Efficiency vs. Effectiveness

Efficiency vs. Effectiveness

- **Efficiency** - účinnost
  - Dosahujeme cílů s minimálním plýtváním zdroji?
  - Kolik prostředků potřebujeme k tomu, co děláme?
  - Používáme “the right tool for the job”?

Efficiency vs. Effectiveness

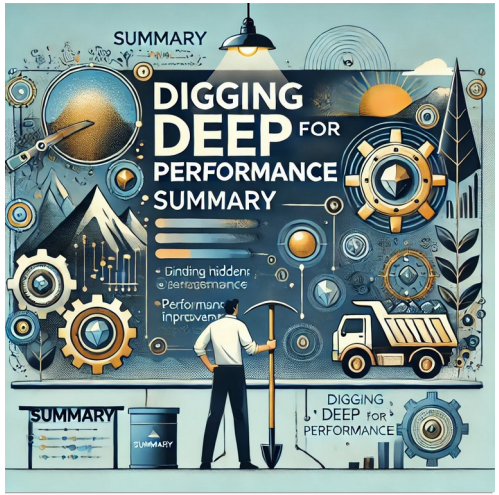
- **Efficiency** - účinnost
  - Dosahujeme cílů s minimálním plýtváním zdroji?
  - Kolik prostředků potřebujeme k tomu, co děláme?
  - Používáme “the right tool for the job”?

- **Effectiveness** - účelnost
  - Pracujeme na těch správných věcech?
  - Neděláme něco zbytečně složitě?
  - Opravdu vše, co děláme, plní svůj účel?

Summary


SUMMARY
DIGGING
DEEP FOR
PERFORMANCE
SUMMARY
SUMMARY
DIGGING
DEEP FOR
PERFORMANCE

Summary



- Vysvětlení pojmů na příkladech
  - Latence vs. Propustnost
  - Memory hierarchy a Data oriented design
  - Branch Predictor a Branchless programming
  - Dynamický vs. Statický Polymorfismus
  - Účinnost vs. Účelnost

- Složitosti dnešního HW … nelze spoléhat na …
  - … to, že CPU se bude chovat adekvátně pro váš use-case
  - … zastaralé poučky (*O(N)* lepší než *O(1)* s vysokým faktorem)
  - … optimalizace překladače (je třeba mít intuici, jak HW funguje)
  - … vlastní intuici (je třeba měřit)
  - … jeden druh měření (je třeba kombinovat různé techniky)

Links & Sources

- **Qminers** ... https://qminers.com

- **Videos**
  - Scott Meyers: Cpu Caches and Why You Care ... https://www.youtube.com/watch?v=WDIkqP4JbkE
  - Mike Acton: Data-Oriented Design and C++ ... https://www.youtube.com/watch?v=rX0ItVEVjHc
  - Matt Godbolt: Compiler Explorer 2023: What's New? ... https://www.youtube.com/watch?v=Ey0H79z_pco
  - Fedor Pikus: Branchless Programming in C++ ... https://www.youtube.com/watch?v=g-WPhYREFjk
  - Fedor Pikus: C++ Type Erasure Demystified ... https://www.youtube.com/watch?v=p-qaf6OS_f4
  - Mathieu Ropert: Data Oriented Design and ECS Explained ... https://www.youtube.com/watch?v=xm4AQj5PHT4
- **Tools**
  - Compiler explorer ... https://godbolt.org
  - Valgrind ... https://valgrind.org/
  - Linux perf ... https://perfwiki.github.io/main
  - Flamegraph visualization ... https://github.com/brendangregg/FlameGraph
  - Optick Profiler for games ... https://optick.dev/
  - Intel VTune Profiler ... https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html
- **Articles**
  - Dan Luu: Branch prediction ... https://danluu.com/branch-prediction/
  - Ulrich Drepper: What every programmer should know about memory ... https://people.freebsd.org/~lstewart/articles/cpumemory.pdf
  - Jobin Johnson: Branchless programming. Does it really matter? ... https://dev.to/jobinrjohnson/branchless-programming-does-it-really-matter-20j4
  - PACMan: Prefetch-Aware Cache Management for high performance caching ... https://ieeexplore.ieee.org/document/7851493
  - C++ Tutorial: Intro to Hash Tables ... https://pumpkinprogrammerdotcom4.wordpress.com/2014/06/21/c-tutorial-intro-to-hash-tables
  - C++ benchmark – std::vector vs. std::list vs. std::deque ... https://baptiste-wicht.com/posts/2012/12/cpp-benchmark-vector-list-deque.html
  - Xaktly - Determinant & Cramer's rule ... https://xaktly.com/MatrixDeterminant.html
  - Profile Guided Optimization (PGO) – Under the Hood ... https://devblogs.microsoft.com/cppblog/profile-guided-optimization-pgo-under-the-hood
  - Björn Fahller – Performance of flat maps ... https://playfulprogramming.blogspot.com/2017/08/performance-of-flat-maps.html

Source: ChatGPT

Qminers®

Děkuji za pozornost

Bonus: Data Oriented Design

Bonus: Data Oriented Design

- **Mapa jako pole**
  - Klíče jsou tříděné
  - Binární nebo sekvenční vyhledávání

Bonus: Data Oriented Design

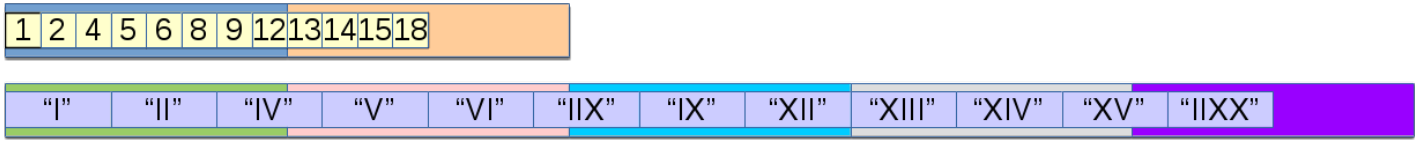
- **Mapa jako pole**
  - Klíče jsou tříděné
  - Binární nebo sekvenční vyhledávání
- **Pole párů**

Credit: *Björn Fahller – Performance of flat maps*

Bonus: Data Oriented Design

- Mapa jako pole
  - Klíče jsou tříděné
  - Binární nebo sekvenční vyhledávání
- Pole párů

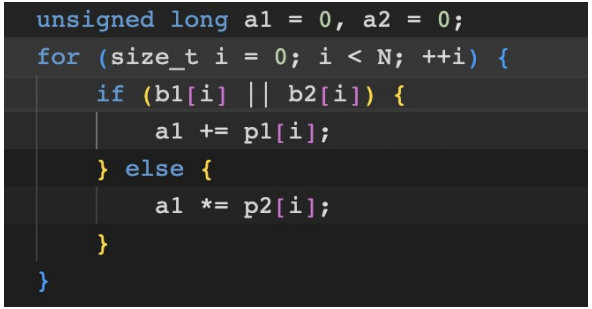
- Dvě oddělená pole



Credit: *Björn Fahller – Performance of flat maps*

Bonus: Branch Prediction

Bonus: Branch Prediction



```
unsigned long a1 = 0, a2 = 0;
for (size_t i = 0; i < N; ++i) {
    if (b1[i] || b2[i]) {
        a1 += p1[i];
    } else {
        a1 *= p2[i];
    }
}
```

Godbolt example

(QuickBench)

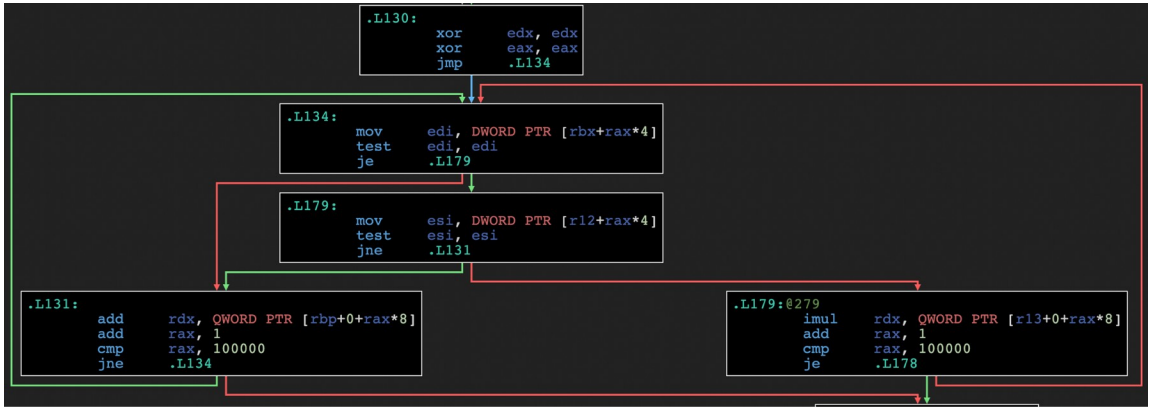
Credit: *Fedor Pikus: Branchless Programming in C++*

Bonus: Branch Prediction

```cpp
unsigned long a1 = 0, a2 = 0;
for (size_t i = 0; i < N; ++i) {
    if (b1[i] || b2[i]) {
        a1 += p1[i];
    } else {
        a1 *= p2[i];
    }
}
```

Godbolt example

(QuickBench)



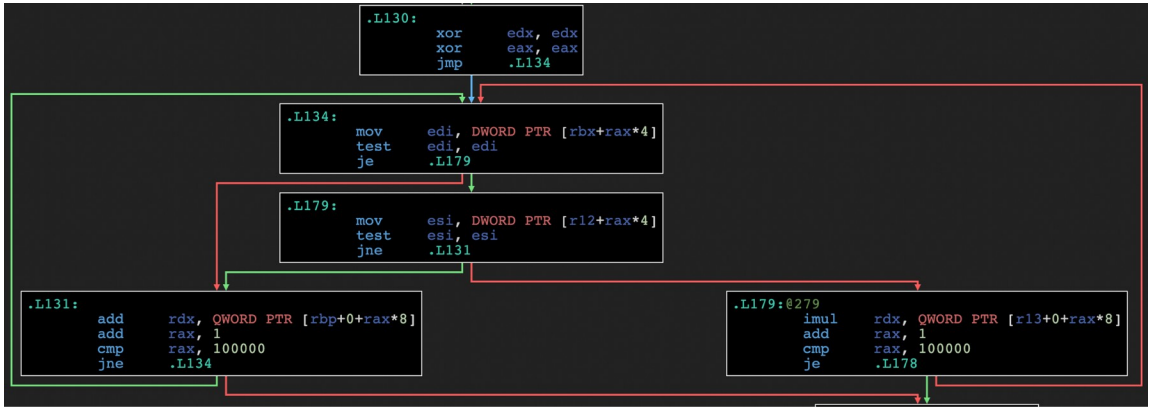
Credit: *Fedor Pikus: Branchless Programming in C++*

Bonus: Branch Prediction

```
unsigned long a1 = 0, a2 = 0;
for (size_t i = 0; i < N; ++i) {
    if (b1[i] || b2[i]) {
        a1 += p1[i];
    } else {
        a1 *= p2[i];
    }
}
```

Godbolt example

(QuickBench)




```
unsigned long a1 = 0, a2 = 0;
for (size_t i = 0; i < N; ++i) {
    if (b1[i] | b2[i]) {
        a1 += p1[i];
    } else {
        a1 *= p2[i];
    }
}
```

Credit: *Fedor Pikus: Branchless Programming in C++*

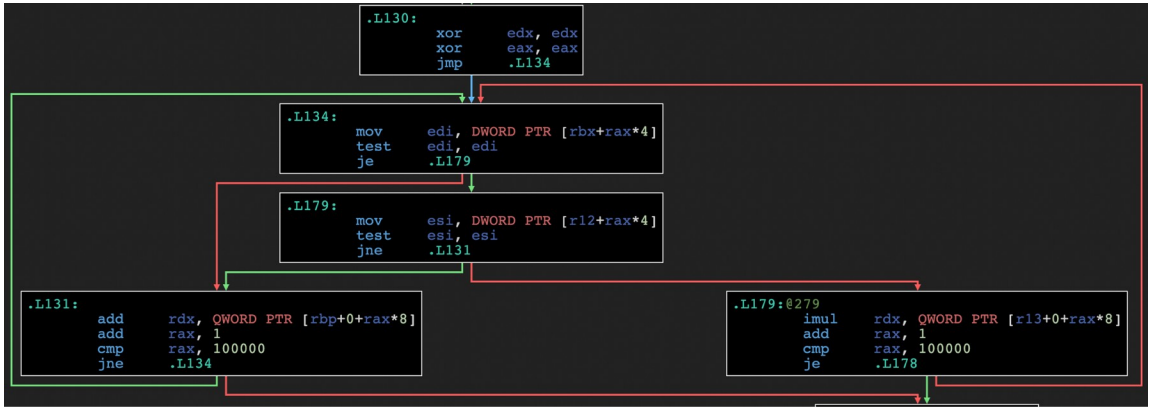
Bonus: Branch Prediction

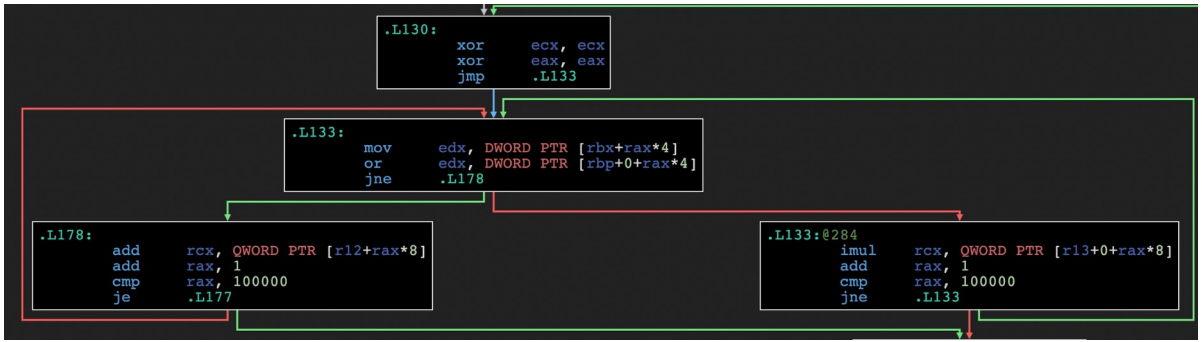
```
unsigned long a1 = 0, a2 = 0;
for (size_t i = 0; i < N; ++i) {
    if (b1[i] || b2[i]) {
        a1 += p1[i];
    } else {
        a1 *= p2[i];
    }
}
```

Godbolt example

(QuickBench)

```
unsigned long a1 = 0, a2 = 0;
for (size_t i = 0; i < N; ++i) {
    if (b1[i] | b2[i]) {
        a1 += p1[i];
    } else {
        a1 *= p2[i];
    }
}
```





Credit: *Fedor Pikus: Branchless Programming in C++*

Bonus: Branch Prediction

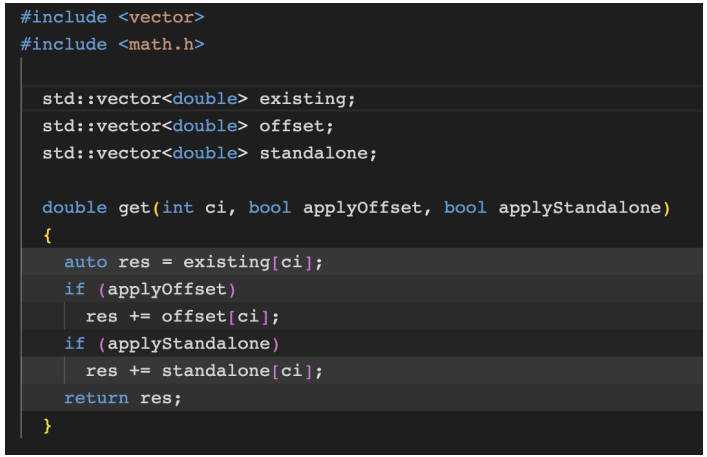
Qminers

	Callgrind (#instr)	Perf (time)	Details
Logical OR	134,563,029	0.853 ms	
Bitwise OR	172,902,709	0.506 ms	

```
-----------------------------------------------------------------------------
Benchmark                   Time             CPU   Iterations UserCounters...
-----------------------------------------------------------------------------
BranchMisprediction     853273 ns       852790 ns          821 items_per_second=117.262M/s

 Performance counter stats for './01-mispredict':

        829.819888      task-clock (msec)         #    0.975 CPUs utilized
                20      context-switches          #    0.024 K/sec
                 1      cpu-migrations            #    0.001 K/sec
              2035      page-faults               #    0.002 M/sec
        2312066437      cycles                    #    2.786 GHz
   <not supported>      stalled-cycles-frontend
   <not supported>      stalled-cycles-backend
        2235067958      instructions              #    0.97  insns per cycle
         353769199      branches                  #  426.320 M/sec
          46081671      branch-misses             #   13.03% of all branches
-----------------------------------------------------------------------------
Benchmark                   Time             CPU   Iterations UserCounters...
-----------------------------------------------------------------------------
BranchPrediction        506101 ns       502065 ns         1383 items_per_second=199.177M/s

 Performance counter stats for './01-predict':

        786.186525      task-clock (msec)         #    0.966 CPUs utilized
                78      context-switches          #    0.099 K/sec
                 1      cpu-migrations            #    0.001 K/sec
              2035      page-faults               #    0.003 M/sec
        2189915718      cycles                    #    2.785 GHz
   <not supported>      stalled-cycles-frontend
   <not supported>      stalled-cycles-backend
        3870538861      instructions              #    1.77  insns per cycle
         475422439      branches                  #  604.720 M/sec
             85181      branch-misses             #    0.02% of all branches
```

Bonus: Branchless Programming

Bonus: Branchless Programming



```cpp
#include <vector>
#include <math.h>

std::vector<double> existing;
std::vector<double> offset;
std::vector<double> standalone;

double get(int ci, bool applyOffset, bool applyStandalone)
{
  auto res = existing[ci];
  if (applyOffset)
    res += offset[ci];
  if (applyStandalone)
    res += standalone[ci];
  return res;
}
```

Bonus: Branchless Programming

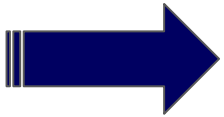

```cpp
#include <vector>
#include <math.h>

std::vector<double> existing;
std::vector<double> offset;
std::vector<double> standalone;

double get(int ci, bool applyOffset, bool applyStandalone)
{
  auto res = existing[ci];
  if (applyOffset)
    res += offset[ci];
  if (applyStandalone)
    res += standalone[ci];
  return res;
}
```

```cpp
#include <vector>
#include <math.h>

std::vector<double> existing;
std::vector<double> offset;
std::vector<double> standalone;

double get(int ci, bool applyOffset, bool applyStandalone)
{
  return existing[ci] + applyOffset * offset[ci] + applyStandalone * standalone[ci];
}
```

Bonus: Branchless Programming

Qminers®

```cpp
#include <vector>
#include <math.h>

std::vector<double> existing;
std::vector<double> offset;
std::vector<double> standalone;

double get(int ci, bool applyOffset, bool applyStandalone)
{
  auto res = existing[ci];
  if (applyOffset)
    res += offset[ci];
  if (applyStandalone)
    res += standalone[ci];
  return res;
}
```



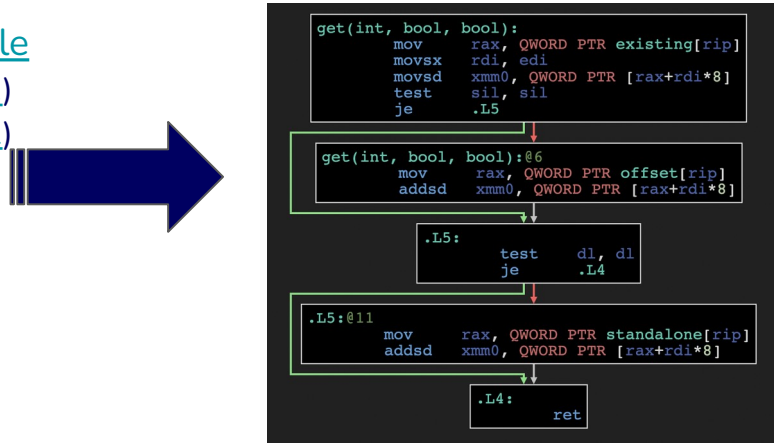
```
get(int, bool, bool):
        mov     rax, QWORD PTR existing[rip]
        movsx   rdi, edi
        movsd   xmm0, QWORD PTR [rax+rdi*8]
        test    sil, sil
        je      .L5
get(int, bool, bool):@6
        mov     rax, QWORD PTR offset[rip]
        addsd   xmm0, QWORD PTR [rax+rdi*8]
.L5:
        test    dl, dl
        je      .L4
.L5:@11
        mov     rax, QWORD PTR standalone[rip]
        addsd   xmm0, QWORD PTR [rax+rdi*8]
.L4:
        ret
```

```cpp
#include <vector>
#include <math.h>

std::vector<double> existing;
std::vector<double> offset;
std::vector<double> standalone;

double get(int ci, bool applyOffset, bool applyStandalone)
{
  return existing[ci] + applyOffset * offset[ci] + applyStandalone * standalone[ci];
}
```

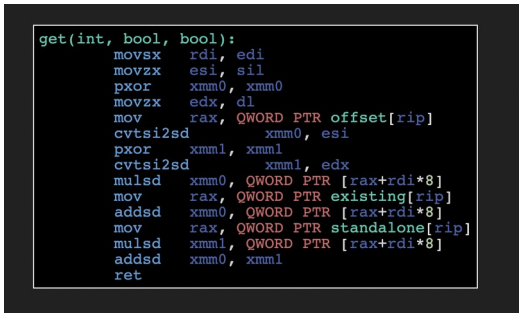
Bonus: Branchless Programming

```cpp
#include <vector>
#include <math.h>

std::vector<double> existing;
std::vector<double> offset;
std::vector<double> standalone;

double get(int ci, bool applyOffset, bool applyStandalone)
{
  auto res = existing[ci];
  if (applyOffset)
    res += offset[ci];
  if (applyStandalone)
    res += standalone[ci];
  return res;
}
```

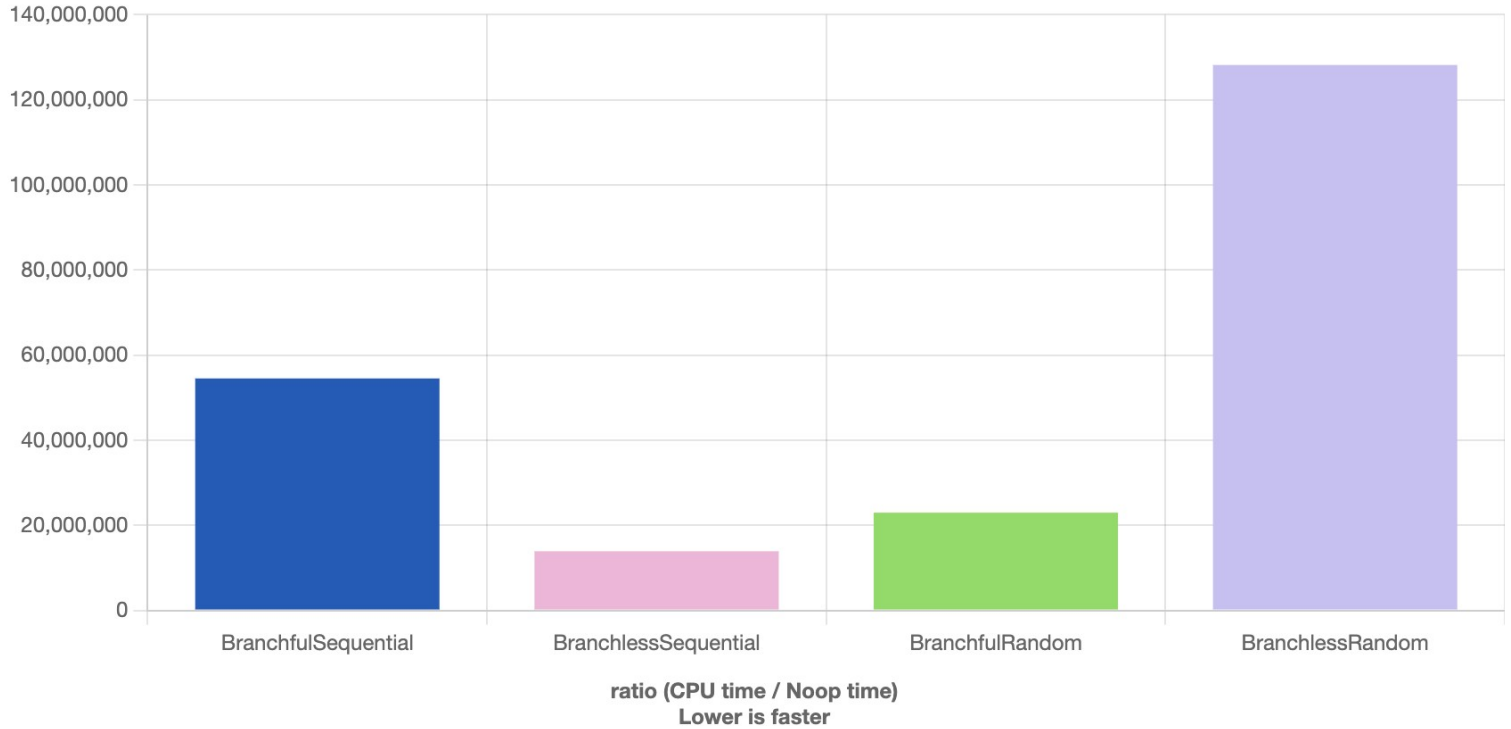
[Godbolt example](#)
([QuickBench #1](#))
([QuickBench #2](#))




```cpp
#include <vector>
#include <math.h>

std::vector<double> existing;
std::vector<double> offset;
std::vector<double> standalone;

double get(int ci, bool applyOffset, bool applyStandalone)
{
  return existing[ci] + applyOffset * offset[ci] + applyStandalone * standalone[ci];
}
```



Bonus: Branchless Programming

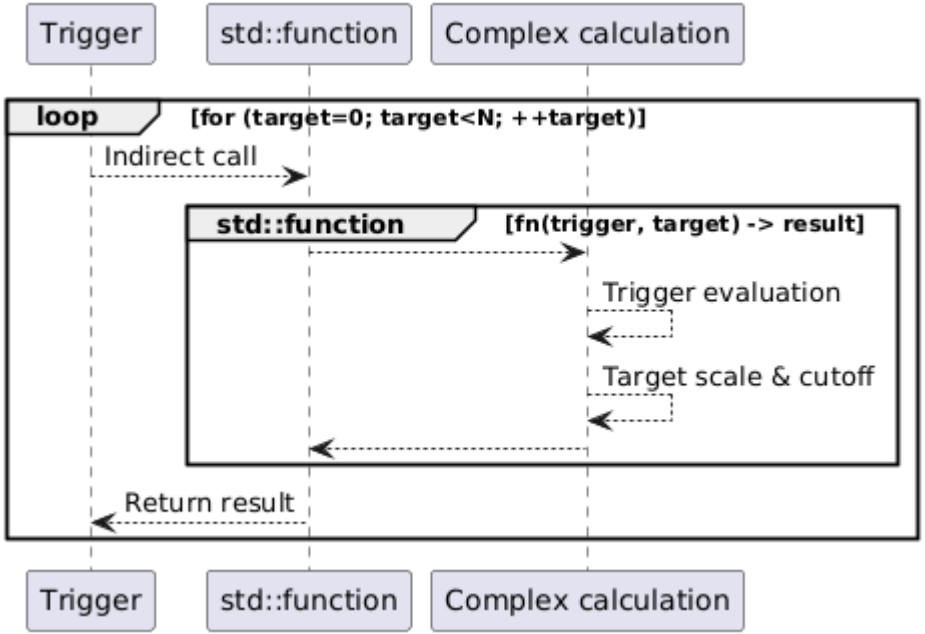


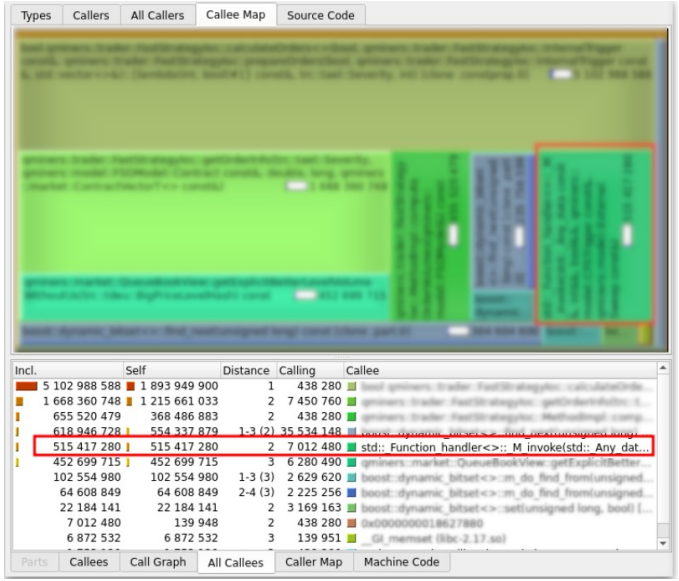
Bonus: Indirect Calls

Bonus: Indirect Calls

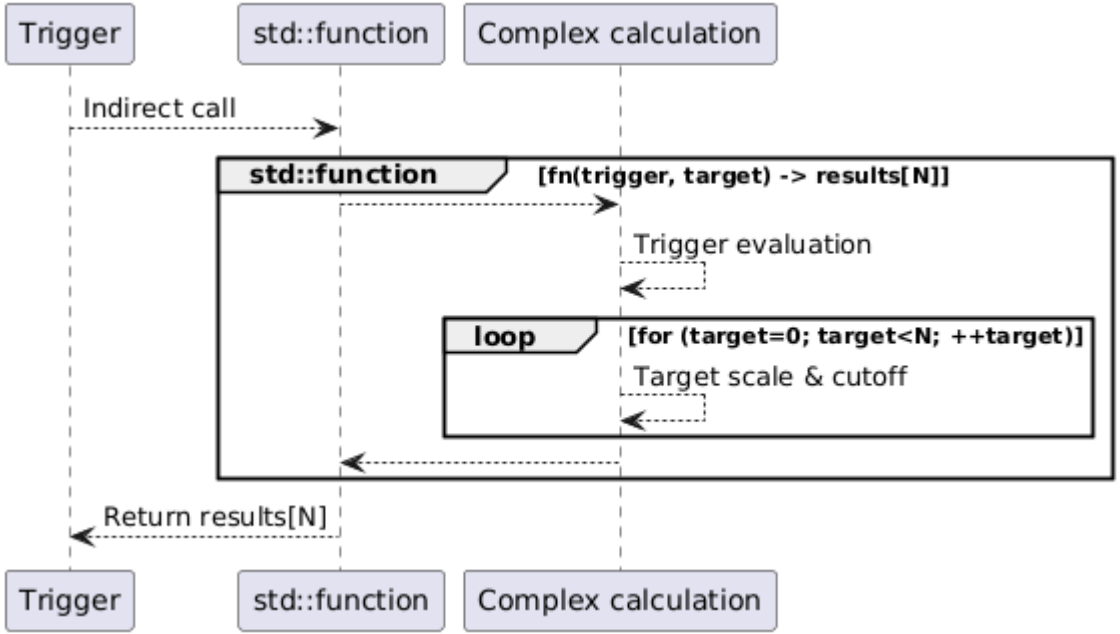
- Nepřímé volání
  - Poměrně drahé
    - Branch misprediction, icache miss, optimalizace
    - Snažíme se mu vyhnout (v kritickém kódu)
  - Jen když ho opravdu potřebujeme
    - “inheritance” (*interface*)
    - “type erasure” (*std::function*)
- Správná granularita
  - Kód je třeba dobře strukturovat
  - Měřit
  - Experimentovat

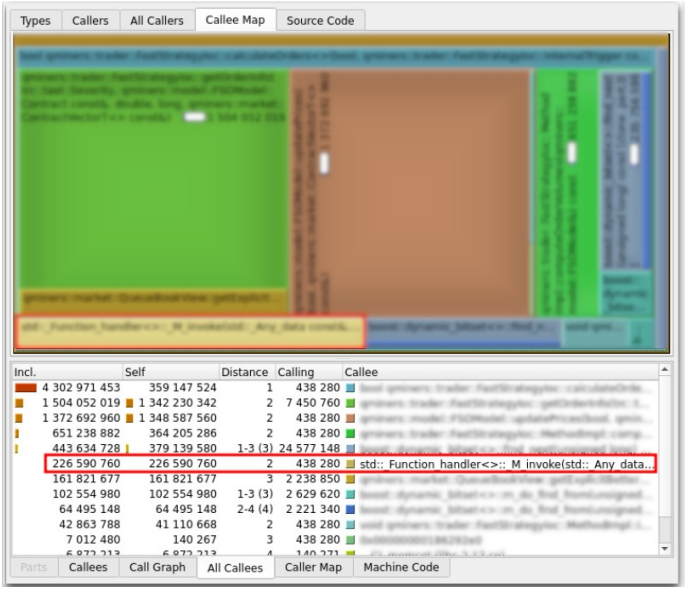
Bonus: Indirect Calls





Bonus: Indirect Calls

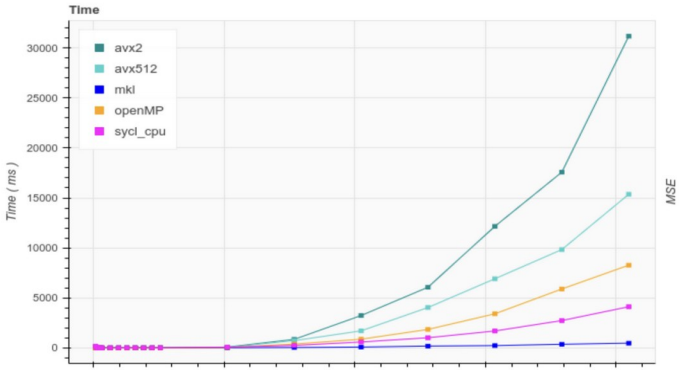




Bonus: Efficiency vs. Effectiveness

Bonus: Efficiency vs. Effectiveness

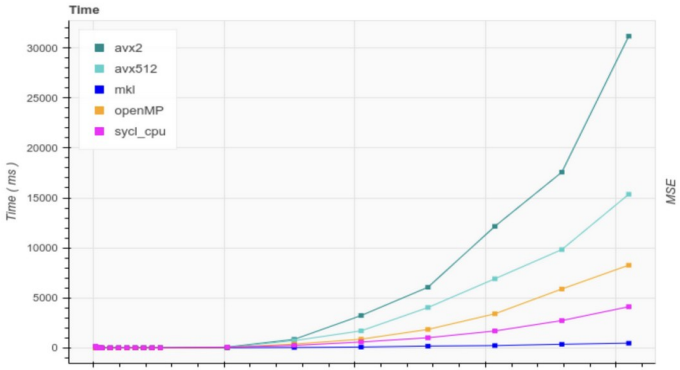
1) **Lineární algebra**
   a) Použití MKL (Math Kernel Library)
   b) Malé matice a vektory, rozklad na block-diagonal-matrix

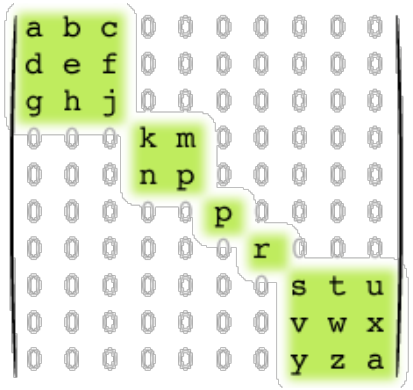


Bonus: Efficiency vs. Effectiveness

Qminers

1) **Lineární algebra**
   a) Použití MKL (Math Kernel Library)
   b) Malé matice a vektory, rozklad na block-diagonal-matrix

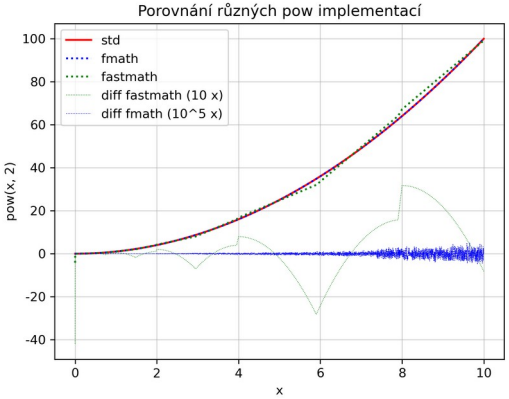


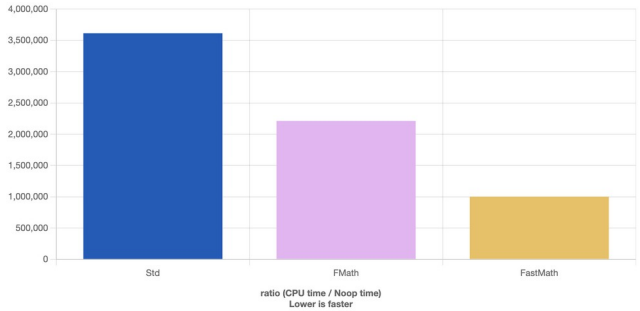


Bonus: Efficiency vs. Effectiveness

Qminers®

1) **Lineární algebra**
   a) Použití MKL (Math Kernel Library)
   b) Malé matice a vektory, rozklad na block-diagonal-matrix
2) **Exponenciální funkce**
   a) Použití knihoven (std, fmath, fastmath, ...)
   b) Je nutné volat *pow(x, alpha)*, když *alpha == 1.05*?



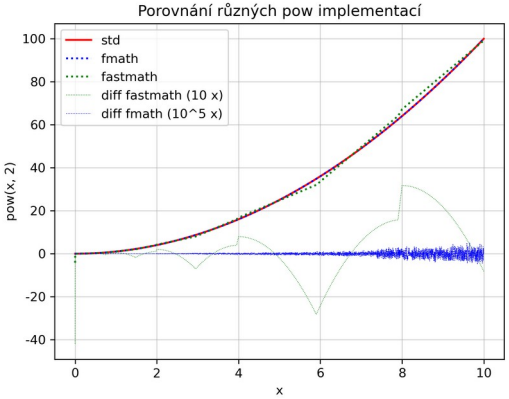


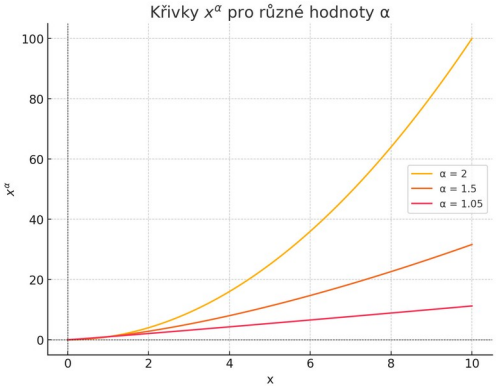
Godbolt example

Bonus: Efficiency vs. Effectiveness

Qminers

1) **Lineární algebra**
   a) Použití MKL (Math Kernel Library)
   b) Malé matice a vektory, rozklad na block-diagonal-matrix
2) **Exponenciální funkce**
   a) Použití knihoven (std, fmath, fastmath, ...)
   b) Je nutné volat *pow(x, alpha)*, když *alpha == 1.05*?





Godbolt example