

Design for Performance

Fedor Pikus

Chief Scientist, Design2Silicon Division
Mentor Graphics (a Siemens business)

Mentor[®]
A Siemens Business

cppcon 
the c++ conference
SEPTEMBER 23-29 2018
Bellevue, Washington, USA

Background

- I design Electronic Design Automation (EDA) tools
 - Used by designers and manufacturers of integrated circuits (IC)
 - Apple, Qualcomm, Intel, TSMC, Samsung, ATI, Nvidia, etc
- Performance is the most important metric after feature capabilities
 - More important than ... anything else?

Customers on correctness

Customer: We want your software to do this and this.

EDA Vendor: We could do it this way but it's not always the right result.

C: OK, we will test it...

... 2 days later ...

C: We ran it on all our designs, it works. Do it!

EV: How many designs is that?

C: Three.

EV: What if it doesn't work on another design?

C: This never happens.

Customers on performance

Customer: How fast is your tool?

EDA Vendor: 12 hours on your largest design, less on others.

C: How many designs is that?

EV: 10. And 20 more from your competitors.

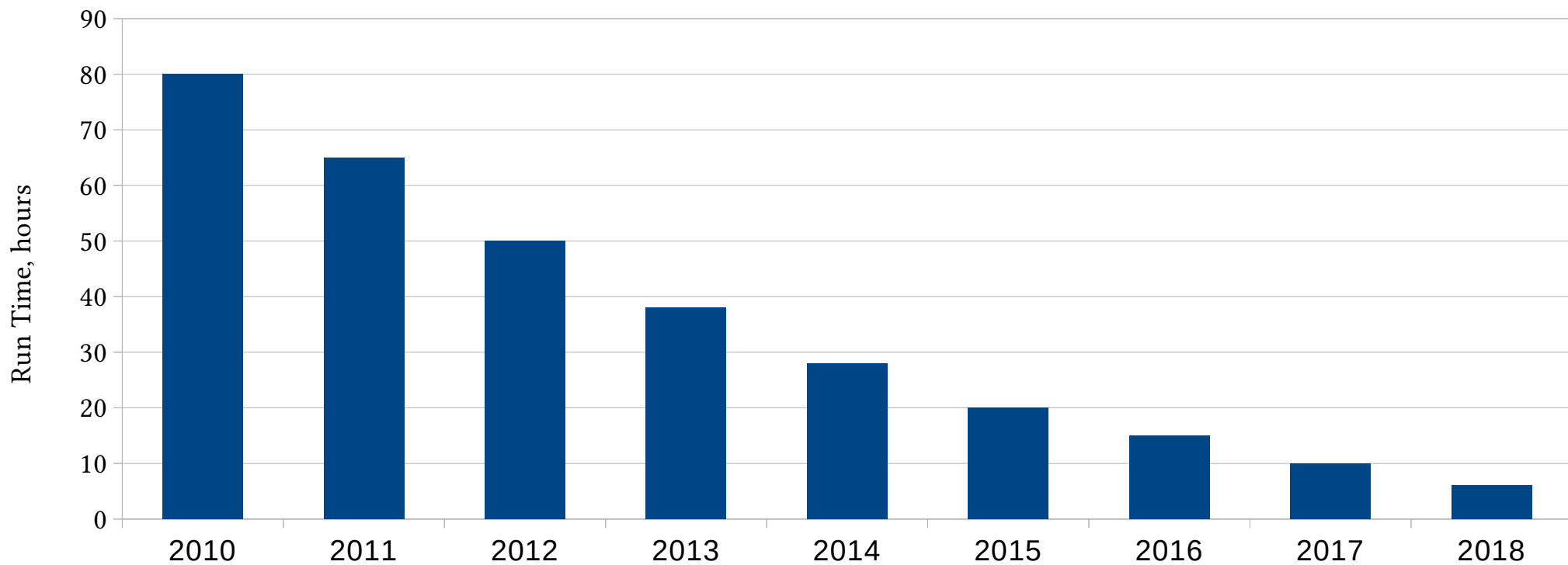
C: What if we get an even larger design?

EV: On all your designs and other designs it runs in under 12 hours

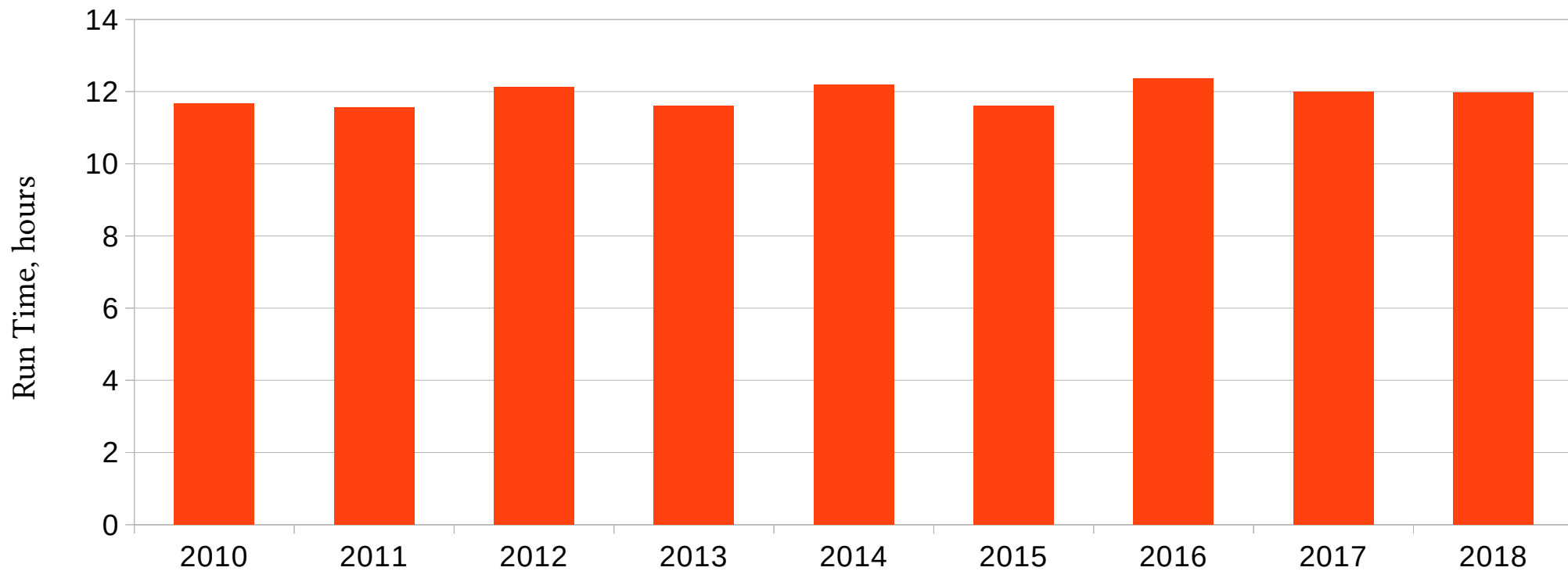
C: But what if some day we get an even larger or a very unusual design?

Write us another tool that can handle even that in 12 hours!

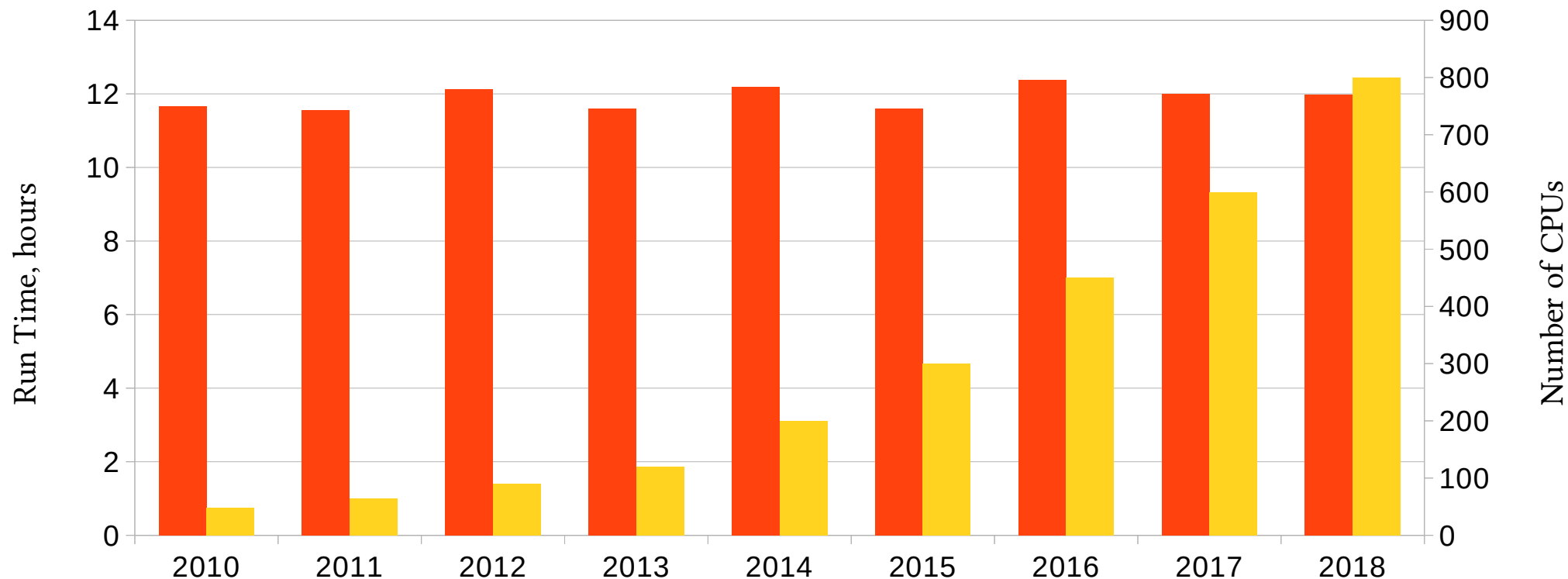
EDA performance progress



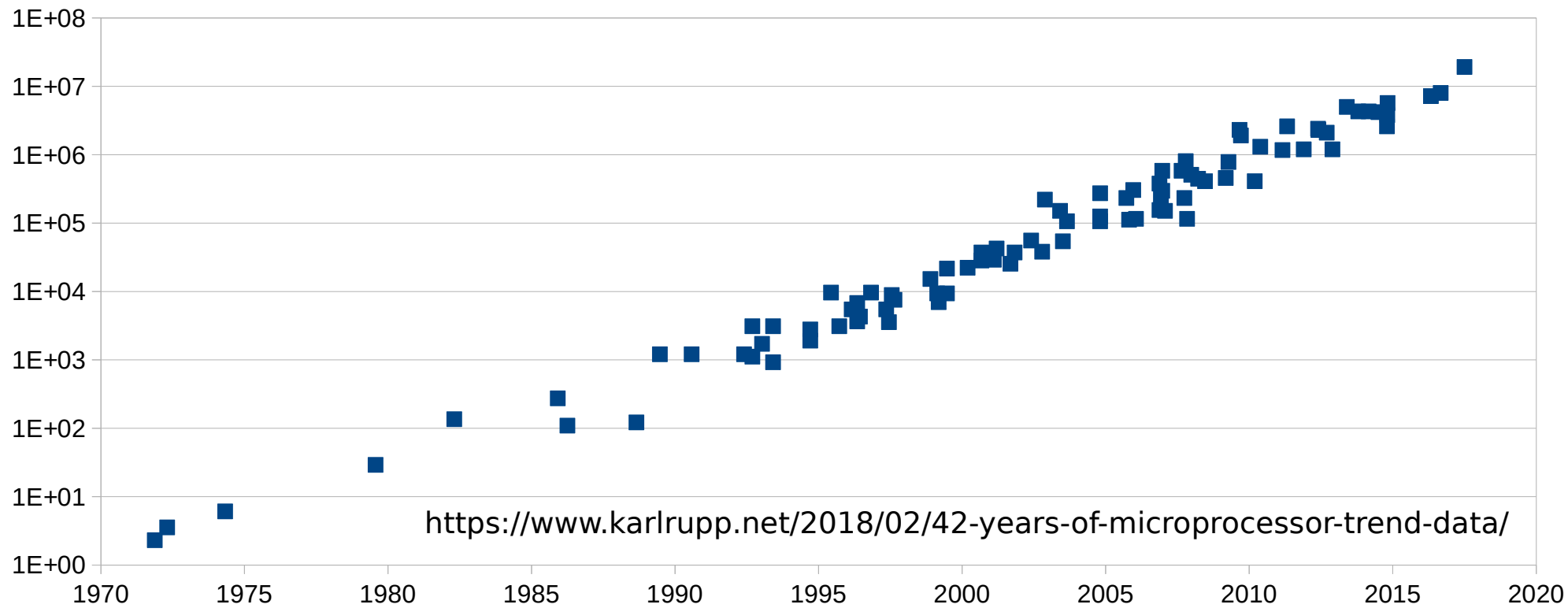
EDA performance on the largest design



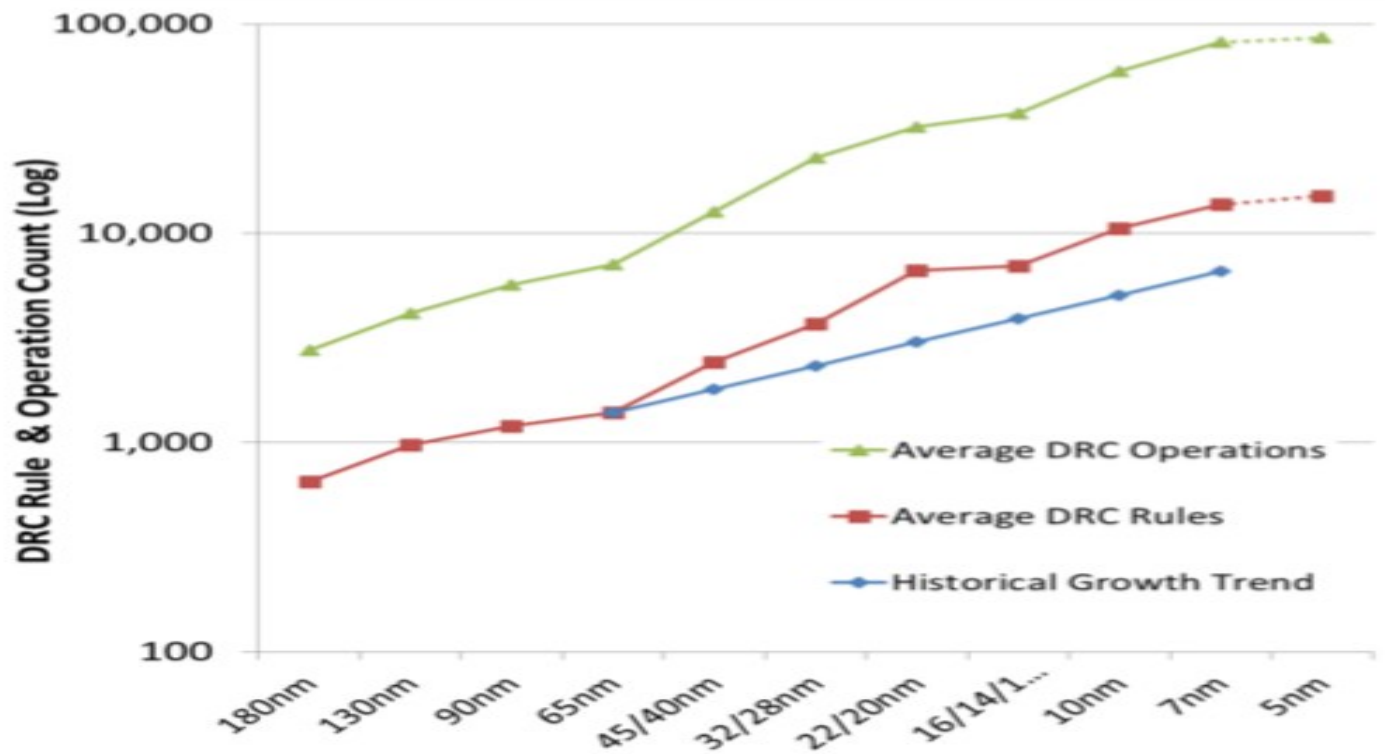
EDA performance on the largest design



Largest design - transistor count



Largest design – complexity



Long-term development and support

Technologies		LVS, DevX, RCX, Rcred	DSPE, Proc	
Mentor Graphics	Calibre	DRC, ERC, LVS, DevX	GDSII, Spice, SVRF, Proc	GDSII, text
	XCalibre	RCX, DevX, RCred, Tim	GDSII, SVRF, Proc	Spice, SPF, SDF, Text

1996

TSMC Adopts Mentor Graphics Calibre Physical Verification Tool Into Its SoC Design Flow

IP News

2000

TSMC Adopts Mentor Graphics Calibre Physical Verification

Mentor launches Calibre RealTime Digital to help cut weeks off of IC design signoff

2018

Mentor, a Siemens business, today announced Calibre® RealTime Digital – a new physical verification tool that works in concert with popular commercial place-and-route environments to ensure “Correct-by-Calibre” routing, and help design teams cut weeks off of IC signoff.

Software design == Survival

- Current crisis can be handled by skillful implementation
 - Often a skillful hack
- Solving immediate problems as they arise accumulates technical debt that must be paid off later
 - Continued development becomes progressively more expensive
 - Maintenance becomes more difficult and uncertain
- Software evolves over time, but usually gradually
 - Complete rewrite takes the tool off market for the duration

Design is the foundation of software

- Design that lasts
- Adaptable to changes
 - Beyond what was expected
- Architecture of some products is the same after 10-15 years
 - A lot of features added
 - Some implementations rewritten
 - Components and their interactions, interfaces, data flows remain



Design is the foundation of software

- Not necessarily **good** software
- Adaptable to changes
 - Not always organized
- Architecture of some products is the same after 10 years
 - Just as chaotic as it was then
- Usually a crisis is needed to justify the necessary rewrite



Design is the foundation of software

- Software is NOT like a house
 - You can build it without a foundation
- After a while nobody can change anything
- Best case: one day it'll break completely
 - Then we can rewrite it



Does good design help performance?

- We know the guiding principles of a good design
 - Encapsulation, modularity, separation of concerns...
- Do they help to write fast code?

Does good design help performance?

- We know the guiding principles of a good design
 - Encapsulation, modularity, interfaces, separation of concerns...
- Do they help to write fast code?
- Sort of : good design makes it easier to improve implementation
- A very good design can also perform poorly and resist optimization
 - Encapsulation, modularity, interfaces, etc must be designed with the understanding of their effect on performance
 - With a good design, the concern is to not lock in bad performance

Good design vs design for performance

- Performance must be one of the initial design objectives
- Not all otherwise good designs are high-performing
- Not all high-performance designs are otherwise good
- How to design for performance?

Good design vs design for performance

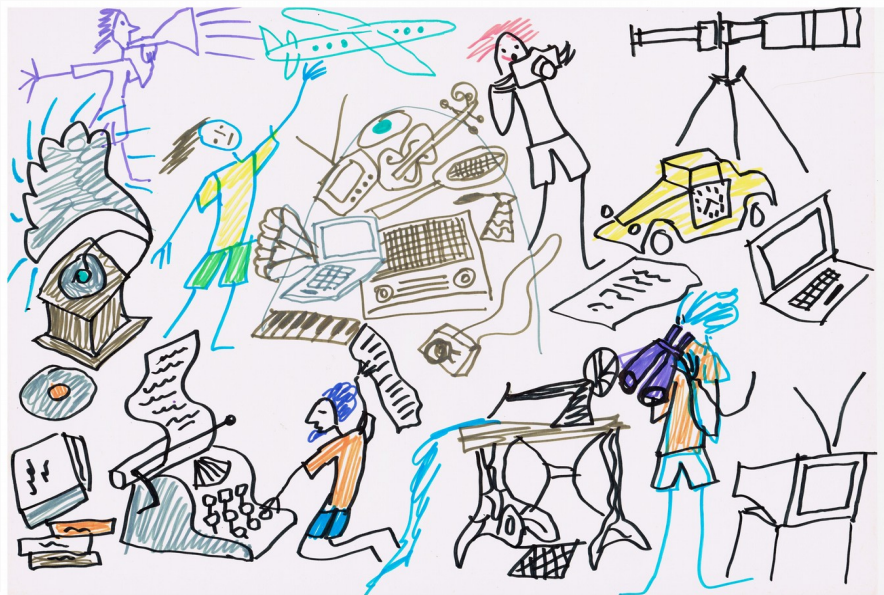
- Performance must be one of the initial design objectives
- Not all otherwise good designs are high-performing
- Not all high-performance designs are otherwise good
- ~~How to design for performance?~~
How to avoid boring talks?



Design: descriptive or prescriptive?

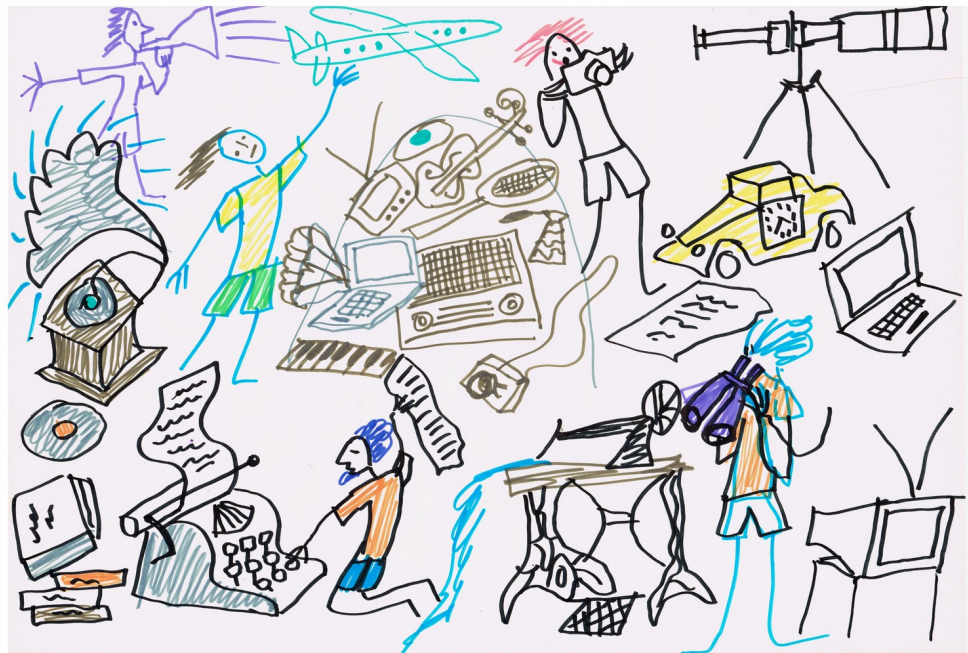
- Designs are like stories

Thanks, Titus!



Good design vs design for performance

- Let's see some code...
- ... and try to make it faster ...
- ... and learn about benchmarking ...
- ... and profiling
- ... and what makes code fast ...
- ... or slow ...
- Along the way, we may discover few things about designing software that are important for performance



What can code teach us about design?

- How can a specific chunk of code help with design?

What can code teach us about design?

- You need to change an old design to improve performance...
Where to start?
- New design is influenced by knowledge about performance
 - A good new design will someday become an old design
- Knowledge about performance has to be gained or confirmed by measurements

What limits performance?

- Different factors can limit performance of a program
- Memory access: memory-bound programs
- CPU speed: computation-bound programs
- Memory use: program performance is limited by the amount of available memory, not by the speed of computation
- Input and output: I/O-bound programs
- Scaling and locking: concurrent programs

What limits performance?

- The distinction is not always clear:
- I/O-bound program that reads and writes compressed data
 - Real limit is the compression/decompression speed
- CPU-bound program with a lot of redundant computations
 - Could have saved intermediate results if we had a lot more memory
- Better scaling means more computations at the same time
 - More computations use more temporary memory, can run out

Code: Matrix-vector product

```
void mv_prod(size_t N, const float* m, const float* v, float* res) {  
    const float* r = m;  
    for (size_t i = 0; i < N; ++i) {  
        float& res1 = res[i];  
        res1 = 0;  
        for (size_t j = 0; j < N; ++j) {  
            res1 += r[j]*v[j];  
        }  
        r += N;  
    }  
}
```

Code: Matrix-vector product

- Is it memory-bound or compute-bound?
- How can we tell?

Code: Matrix-vector product

```
inline float prod8(const float* a, const float* b) {  
    union { float r[8]; __m256 rv; };  
    rv = _mm256_dp_ps(_mm256_load_ps(a),  
                     _mm256_load_ps(b), 0xf1);  
    return r[0] + r[4];  
}
```

Code: Matrix-vector product

```
float vv_prod(size_t N, const float* a, const float* b) {  
    float res = 0;  
    for (size_t i = 0; i < N; i += 8) { res += prod8(a + i, b + i); }  
    return res;  
}
```

Design consideration

- Prefer high-granularity API
 - Restrictive: process one element at a time (make no assumptions that there will be more)
 - Permissive: process all these elements
- Alternative – process one element at a time but finalize explicitly
 - Ewww...

Design consideration

- Aligned memory is needed!
 - Depending on how memory is allocated, could need a major rewrite
 - In general, memory management is a constraint in many designs, needs to be considered early

Code: Matrix-vector product

```
inline float prod8(const float* a, const float* b) {  
    union { float r[8]; __m256 rv; };  
    rv = _mm256_dp_ps(_mm256_load_ps(a),  
                      _mm256_load_ps(b), 0xf1);  
    return r[0] + r[4];  
}
```

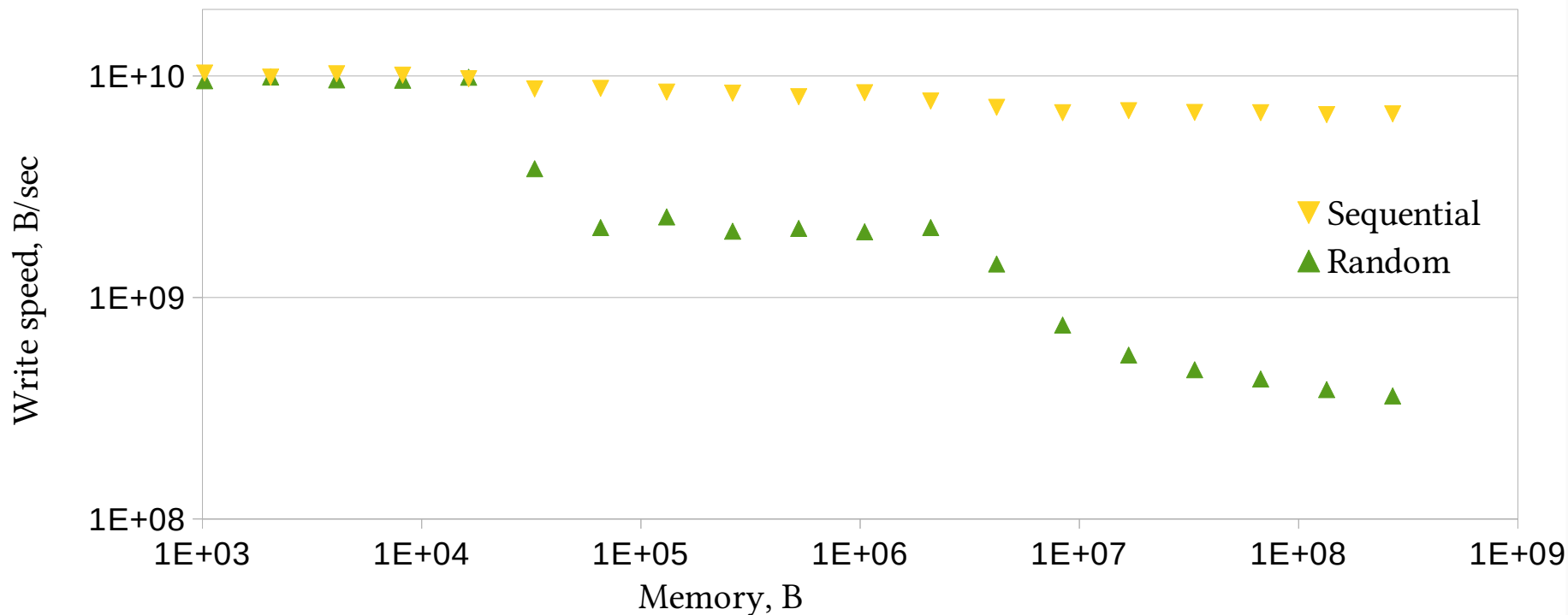
Design consideration

- High-performance code is often low-level
 - Sometimes non-portable
 - Sometimes abuses the language
- Good design contains such C++ code
 - Esoteric knowledge and arcane code contained in a small part of the code
 - Higher level abstractions provided for the rest of the code
 - Take care: abstractions and APIs must not over-constrain the implementation or force overhead

Design consideration

- Memory access speed depends on how memory is accessed
 - Is the accelerated code fast because it computes faster, or because it reads memory faster? Some of both
 - Memory access speed can be a bottleneck, needs more thought

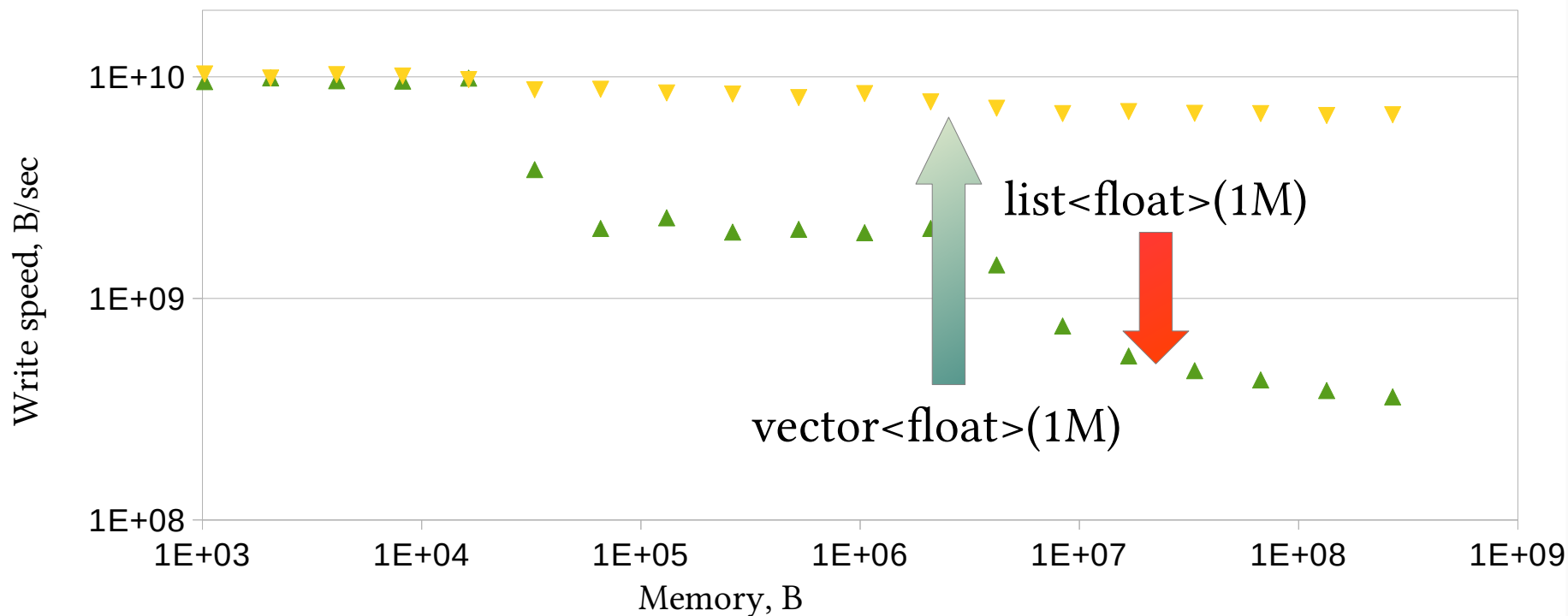
Memory Speed



Design consideration

- Data organization is chosen during design
 - Data organization and data structures determine possible ways to access the data, and their efficiency
 - Ways to access the data determine which algorithms can be used

Memory Speed



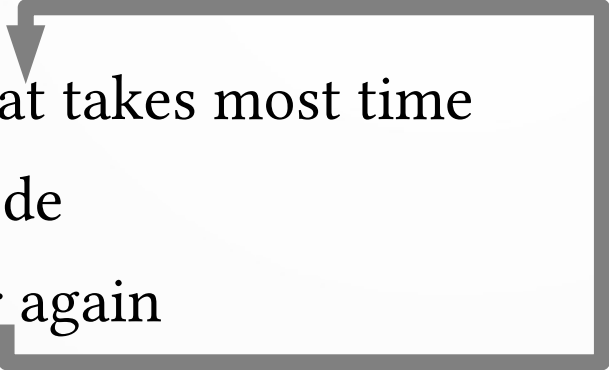
More code

- Can it really be that bad?

More code

- Can it really be that bad?
- It's worse.

Performance made simple

- 1) Run the profiler
 - 2) Find the step that takes most time
 - 3) Optimize the code
 - 4) Run the profiler again
- 



Performance made hard again

- 1) Run the profiler
- 2) Find the step that takes most time
- 3) Optimize the code
- 4) Run the profiler again
- 5) Until nothing stands out
 -) But nothing is very good either
- 6) Can it really happen? And what does it mean?



What to optimize when there is nothing to optimize?

- When there is no hot code, there is hot data!
- Sometimes a single data access function stands out
 - But not its callers, calls are scattered all over
 - The access function itself may be impossible to optimize
 - At least within its current interface contract
- Two options: access less data or organize data differently
 - Both are design issues

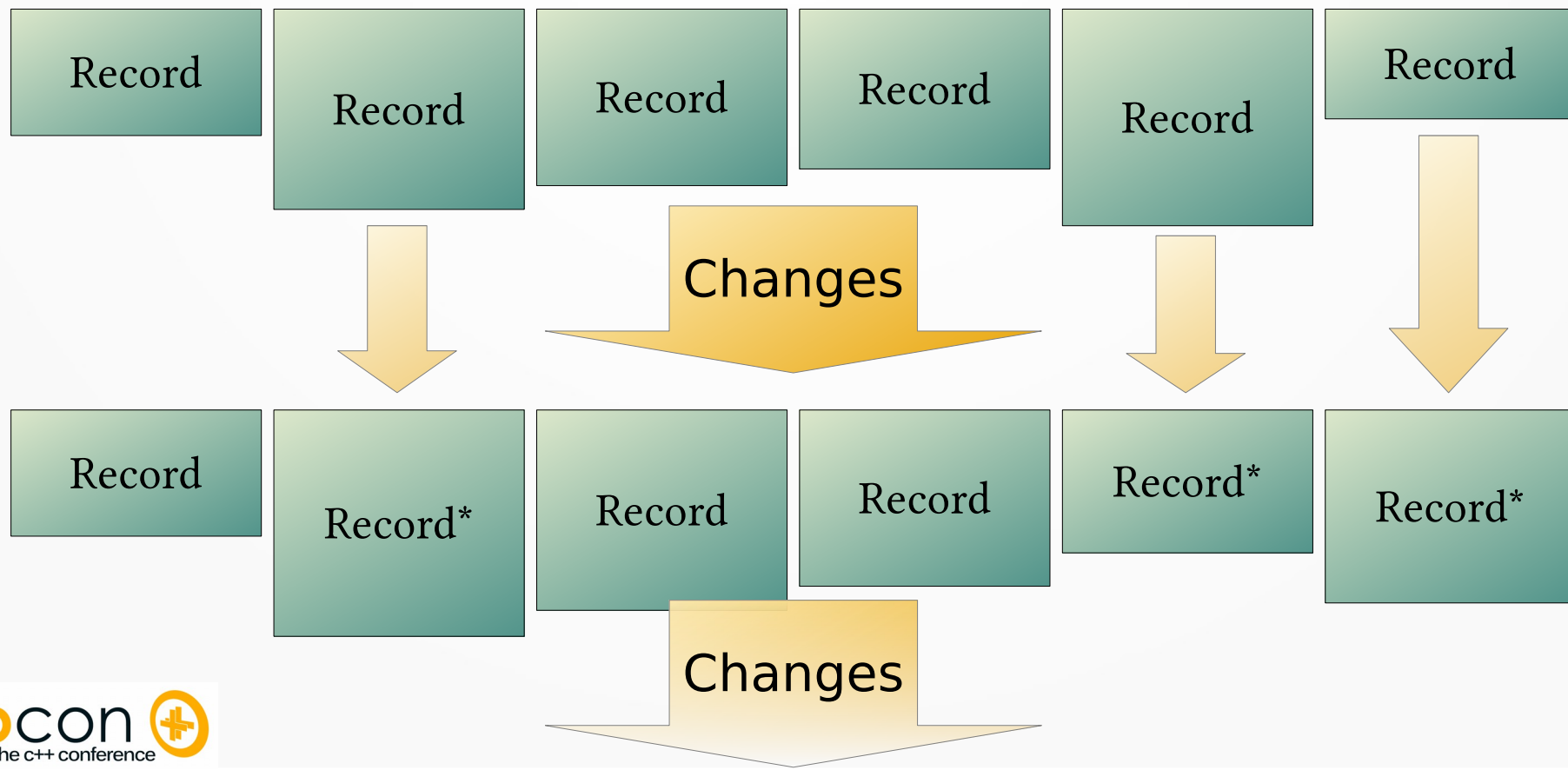
Design considerations

- Design determines the data flow at the high level
 - What are the components of the system?
 - Who has what information, and who needs it?
 - What are the data flows?
- Data organization determines what information can be accessed
- Data access requirements limit performance
 - API which exposes implementation can limit performance

Design considerations - example

- `std::unordered_map` – a hash table
 - It has hashing functions
- `insert()`, `find()`, `erase()` – normal hash table stuff
- Bucket iterators...
 - `begin(n)`, `end(n)`
 - It's a list!
- All I wanted was a hash table, the API restricted the implementation
 - There are implications for performance

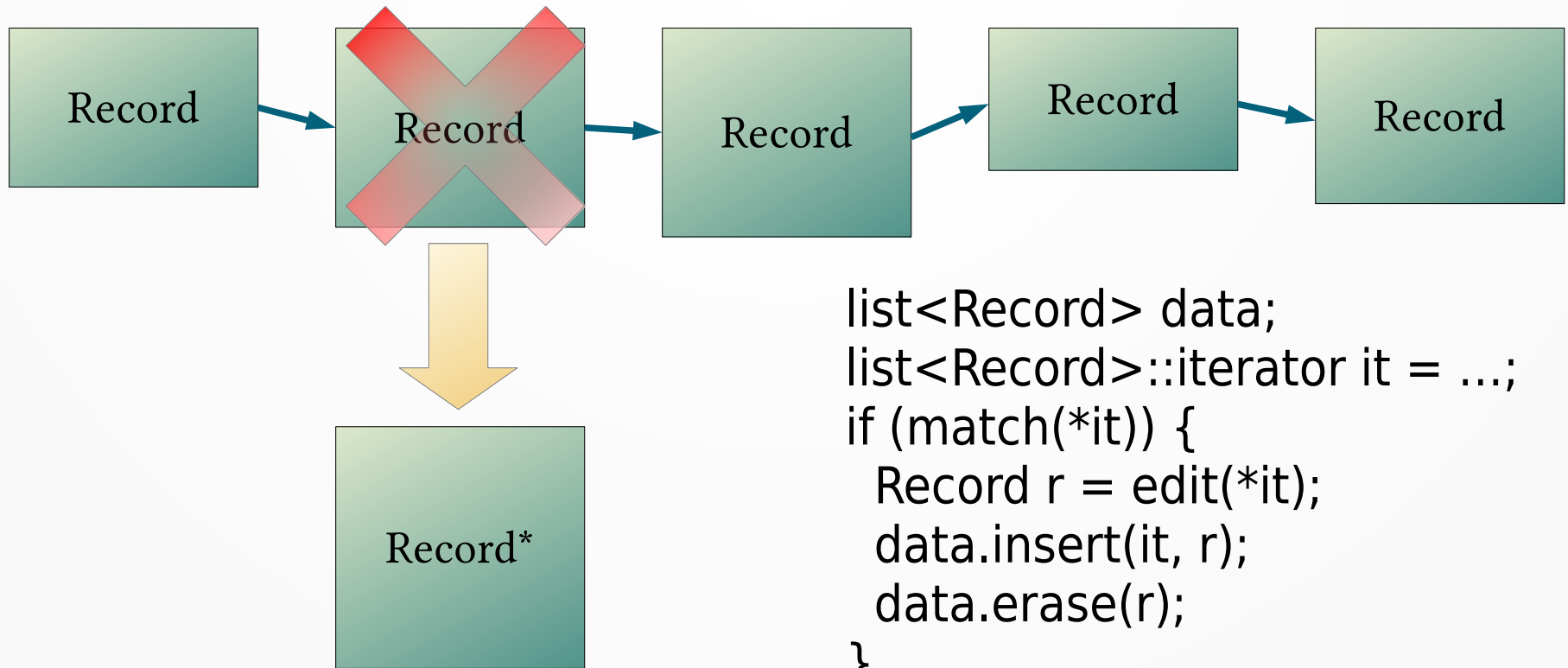
Design example



Design example – random access



Design example – random access

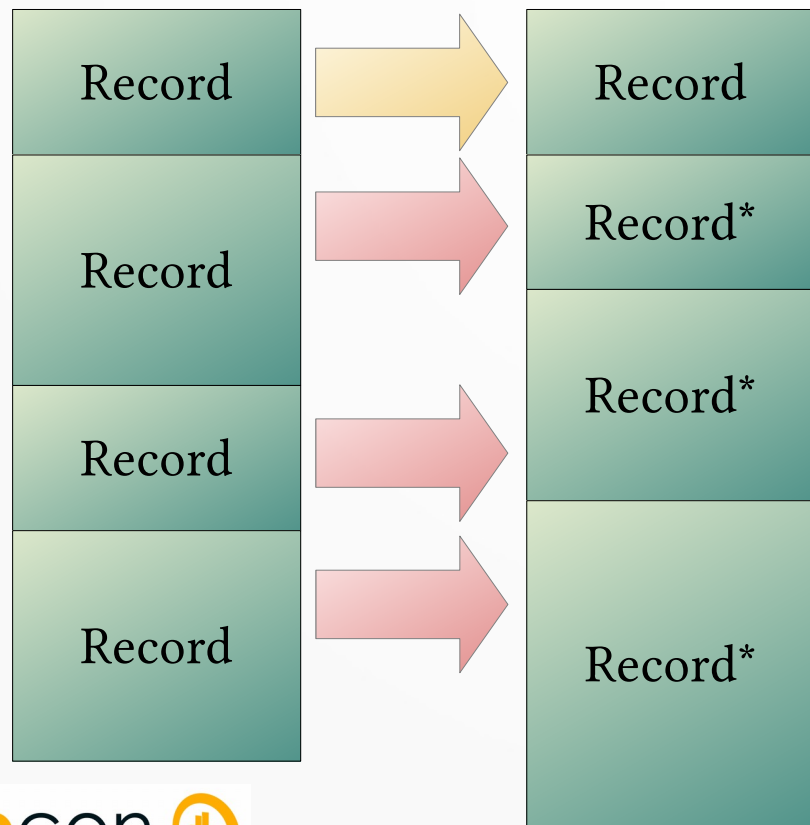


```
list<Record> data;  
list<Record>::iterator it = ...;  
if (match(*it)) {  
    Record r = edit(*it);  
    data.insert(it, r);  
    data.erase(r);  
}
```


Design example – random access

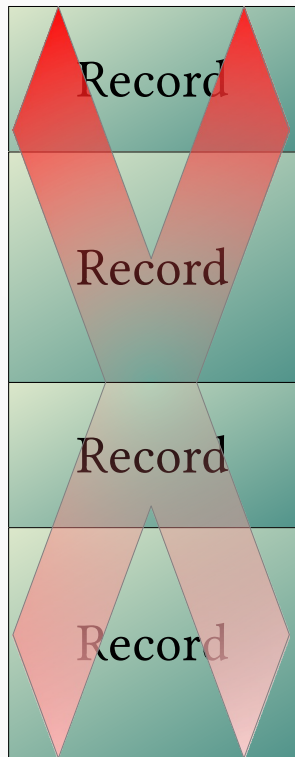


Design example – streaming access



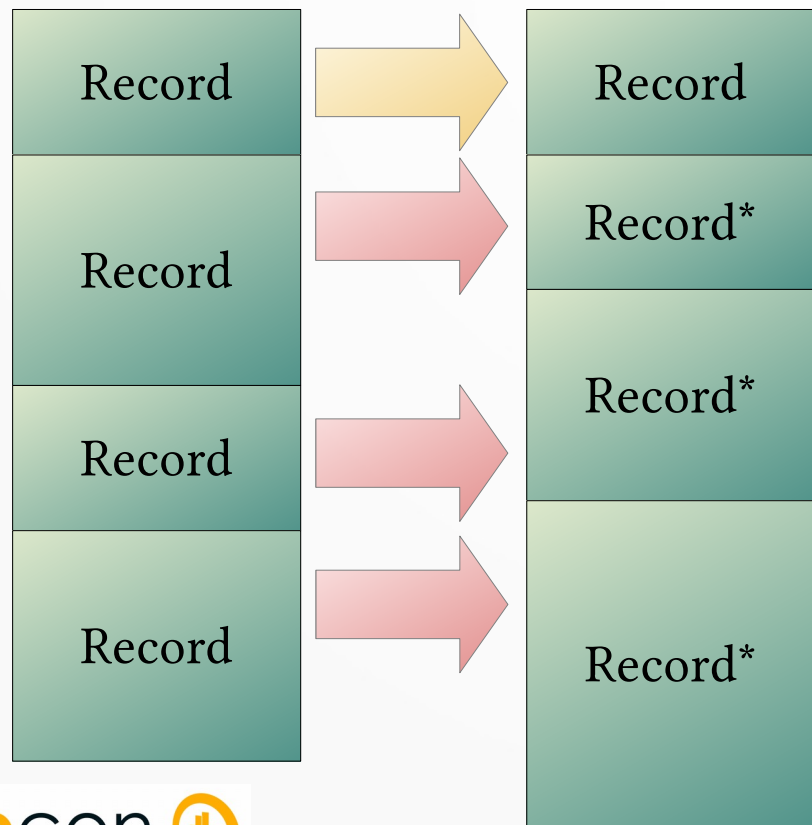
```
Record* data;  
Record* new_data;  
if (match(data)) {  
    *new_data = edit(data);  
} else {  
    *new_data = *data;  
}
```

Design example – streaming access



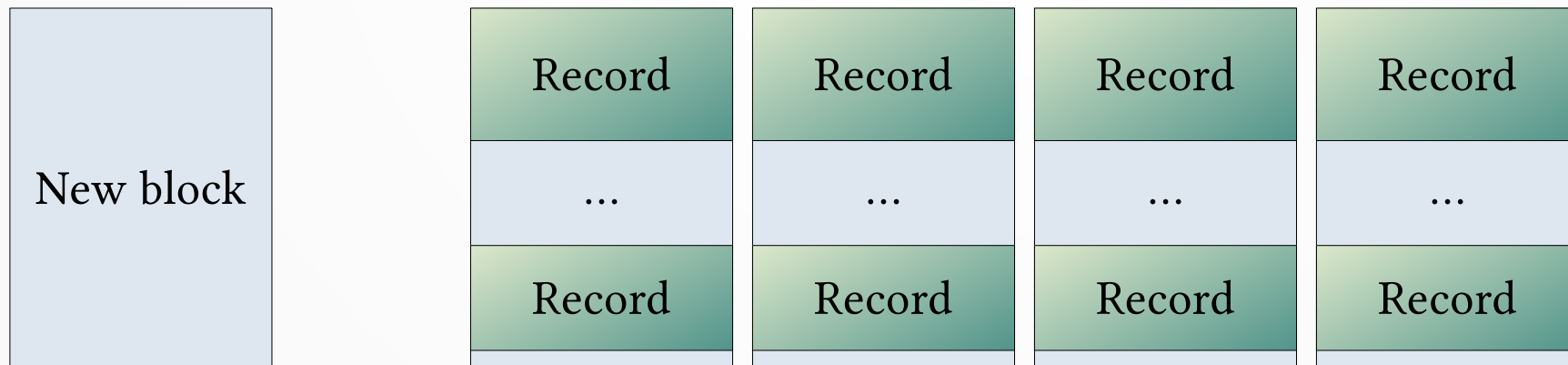
```
Record* data;  
Record* new_data;  
if (match(data)) {  
    *new_data = edit(data);  
} else {  
    *new_data = *data;  
}  
deallocate(data);  
data = new_data;
```

Design example – streaming access

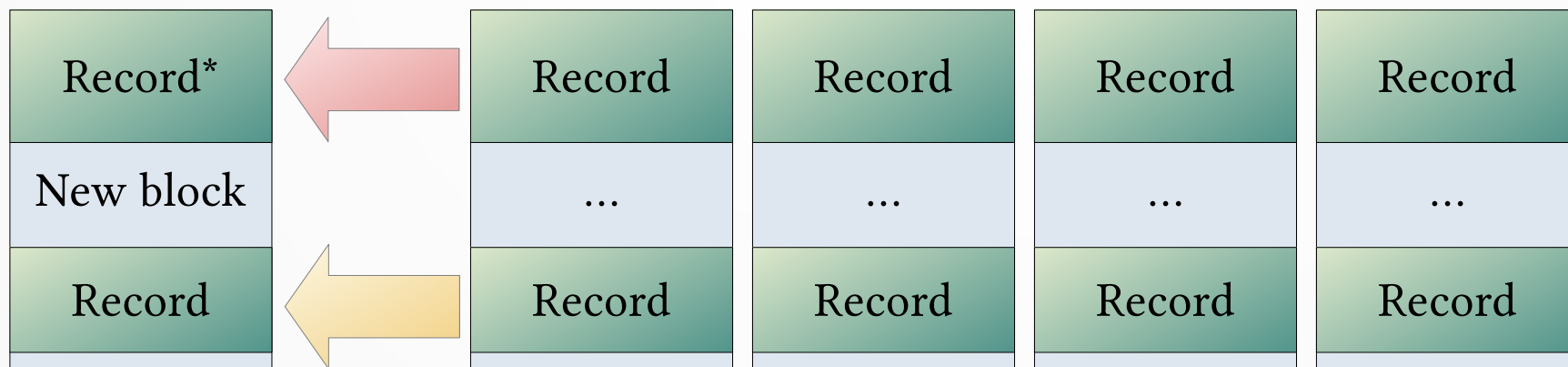


- Consumes twice as much memory at the peak usage
- Unless we can recycle it...

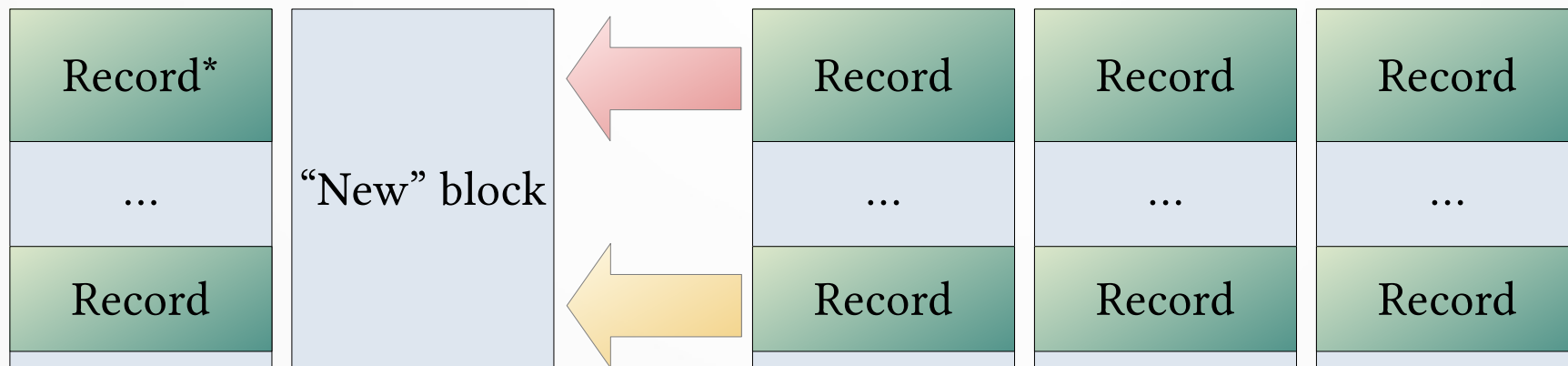
Design example – block streaming access



Design example – block streaming access



Design example – block streaming access



- Extra memory is limited to one block
- Most recently used memory is reused (hot in cache)

Design considerations

- Restricted data access allows for a more efficient implementation
 - Data can be organized for optimal access performance
- Restricting data access also restricts available operations
 - Some operations will need additional memory accesses or copying
 - Performance difference between optimal and random access is huge
- Temporary data conversion to a different format may be required
 - If these conversions are too frequent, a different design is needed

Design considerations

- Design for performance must be influenced by measurements
 - You cannot make good decisions without good knowledge
- Measuring performance requires a working program
 - Implementing a program requires design
- Solution: prototypes, mocks, and experiments
 - Example: memory write speed (just need to know access patterns and data volume, does not need to be real data or real use of data)

Design for performance - examples

- Design for performance must be influenced by measurements
 - Saving temporary data to disk: what is the minimum time to write and read the expected volume of data?
 - Compressing data in memory: how long does it take to decompress?
 - Extensible hierarchy via run-time polymorphism: what is the overhead of the virtual function call?
 - Data organization allows limited access: what is the cost of an extra pass over all data (extra copy, compression/decompression, etc)

Design for concurrency



Undefined Behavior

- Undefined behavior is really a narrow contract
- If the inputs are in contract, correct behavior is guaranteed
- If the inputs are out of contract, anything can happen
- Use undefined behavior to improve performance!

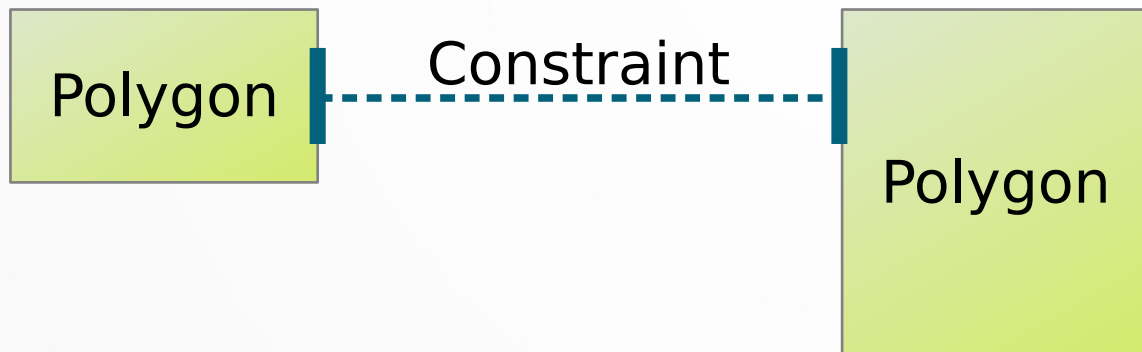
Undefined Behavior

- Undefined behavior is really a narrow contract
- If the inputs are in contract, correct behavior is guaranteed
- If the inputs are out of contract, anything can happen
- Use undefined behavior to improve performance!

```
a[i++] = f(i=10, a=new int[++i]);
```

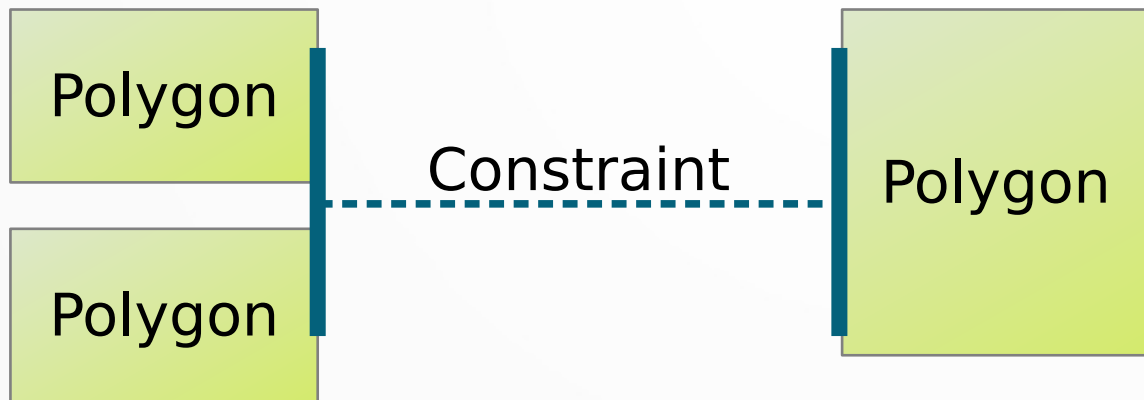
Not this kind of UB!
Ansel & Barbara

Undefined Behavior Example



- Operation: build a constraint graph where polygons are nodes and constraints are edges

Undefined Behavior Example



- Edge connects 2 nodes. Not 3. Undefined behavior!
- Expensive to handle consistently. What really happens?
 - An edge can connect any two nodes. Or none at all.

Undefined Behavior

- Undefined behavior is really a narrow contract
- If the inputs are in contract, correct behavior is guaranteed
- If the inputs are out of contract, anything can happen
- Use narrow contracts to improve performance!
 - Narrow contracts (undefined behavior) allows the implementation to be inconsistent where unnecessary consistency would be expensive
- Provide a way to detect undefined behavior

Design for performance

- Design for performance must be influenced by measurements
- Data organization and information flow limit peak performance
- APIs should not expose or mandate the implementation
- APIs should present logical tasks or transactions
 - Preferably with large granularity
- Contain exotic code in few encapsulated areas of the system
- Good design practices are (mostly) good for performance too
 - Encapsulation, minimizing interactions, modularity
- Use narrow contracts (undefined behavior) to your advantage

Design for Performance

Fedor Pikus

Cartoons by Evgenia Golant