Title: Front End - Mathematica - 4

Date: Sep 02, 2016 09:00 AM

URL: http://pirsa.org/16090040

Abstract:

Algebra of γ Matrices

(Original notebook by Pedro Vieira)

In[2]:=
```
SetAttributes[δ, Orderless];

NumQ[x_?NumericQ] = True;
NumQ[δ[x_, y_]] = True;
NumQ[x_] = False;

rules = {c___ · γ[a_] · γ[b_] · d___ :→ c · (-γ[b] · γ[a] + 2 δ[a, b]) · d /; Not@OrderedQ[{a, b}],
    c___ · γ[a_] · γ[a_] · d___ :→ c · δ[a, a] · d,
    a1___ · a2_ · a3___ :→ a2 a1 · a3 /; NumQ[a2],
    a1___ · (a2_ a4_) · a3___ :→ a2 a1 · a4 · a3 /; NumQ[a2],
    a1___ · (a2_ + a4_) · a3___ :→ a1 · a2 · a3 + a1 · a4 · a3,
    CenterDot[a__] :→ Flatten[CenterDot[a]],
    δ[c_, c_] → dim,
    δ[c_, d_]^2 → dim,
    δ[a1_, a2_] a3_ :→ (a3 /. a1 → a2) /; Not@FreeQ[a3, a1]};
```

In[7]:=
```
γ[d] · γ[a] · γ[c] · γ[b] · γ[d] · γ[c] · γ[a] //. rules /. CenterDot[γ[a_]] → γ[a] //
 Simplify
```

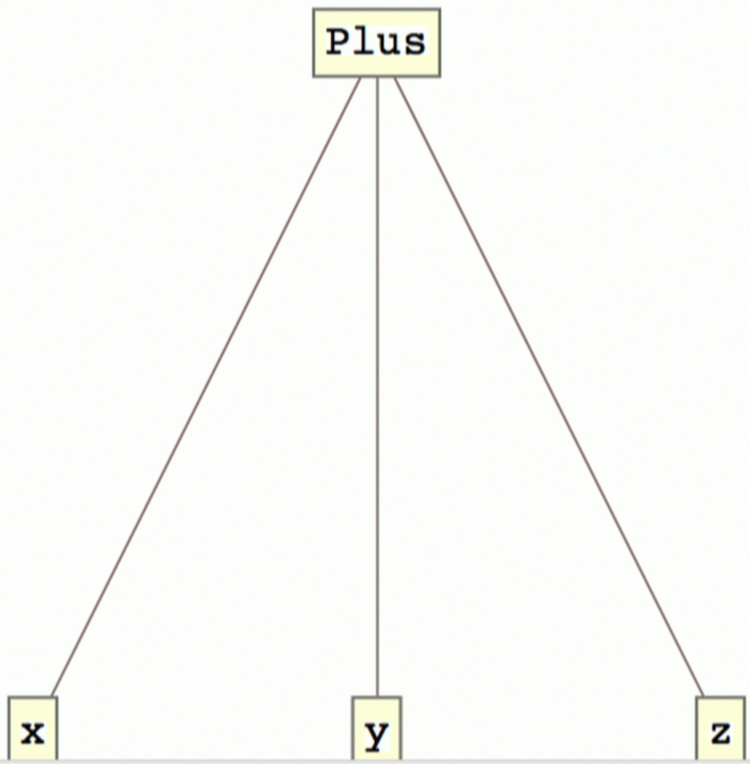
150%

Instead, we will only define how they behave, and what rules can be applied to expressions containing them

We will use three symbols: γ δ CenterDot ·

In[1]:=
```
Plus[x, Plus[y, z]] // TreeForm
```

Out[1]//TreeForm=



150%

tutorial/Attributes

Virtual Book > Core Language > Evaluation of Expressions > Attributes

```
In[6]:= Attributes[f] = {}
Out[6]= {}
```

`Orderless`	orderless, commutative function (arguments are sorted into standard order)
`Flat`	flat, associative function (arguments are "flattened out")
`OneIdentity`	$f[f[a]]$, etc. are equivalent to a for pattern matching
`Listable`	f is automatically "threaded" over lists that appear as arguments (e.g., $f[\{a, b\}]$ becomes $\{f[a], f[b]\}$)
`Constant`	all derivatives of f are zero
`NumericFunction`	f is assumed to have a numerical value when its arguments are numeric quantities
`Protected`	values of f cannot be changed
`Locked`	attributes of f cannot be changed
`ReadProtected`	values of f cannot be read
`HoldFirst`	the first argument of f is not evaluated
`HoldRest`	all but the first argument of f are not evaluated
`HoldAll`	none of the arguments of f are evaluated
`HoldAllComplete`	the arguments of f are treated as completely inert
`NHoldFirst`	the first argument of f is not affected by `N`
`NHoldRest`	all but the first argument of f are not affected by `N`
`NHoldAll`	none of the arguments of f are affected by `N`
`SequenceHold`	`Sequence` objects appearing in the arguments of f are not flattened out
`Temporary`	f is a local variable, removed when no longer used
`Stub`	`Needs` is automatically called if f is ever explicitly input

The complete list of attributes for symbols in the Wolfram Language.

Here are the attributes for the built-in function `Plus`.

```
In[7]:= Attributes[Plus]
Out[7]= {Flat, Listable, NumericFunction, OneIdentity, Orderless, Protected}
```

100%

Goal: Define properties of γ matrices, so that Mathematica can simplify respective expressions

Approach: We will *not* define what γ matrices are, or how they can be represented
Instead, we will only define how they behave, and what rules can be applied to expressions containing them

We will use three symbols: γ δ CenterDot ·

In[2]:=

```
SetAttributes[δ, Orderless];

NumQ[x_?NumericQ] = True;
NumQ[δ[x_, y_]] = True;
NumQ[x_] = False;

rules = {c___ · γ[a_] · γ[b_] · d___ :→ c · (-γ[b] · γ[a] + 2 δ[a, b]) · d /; Not@OrderedQ[{a, b}],
   c___ · γ[a_] · γ[a_] · d___ :→ c · δ[a, a] · d,
   a1___ · a2_ · a3___ :→ a2 a1 · a3 /; NumQ[a2],
   a1___ · (a2_ a4_) · a3___ :→ a2 a1 · a4 · a3 /; NumQ[a2],
   a1___ · (a2_ + a4_) · a3___ :→ a1 · a2 · a3 + a1 · a4 · a3,
   CenterDot[a__] :→ Flatten[CenterDot[a]],
   δ[c_, c_] → dim,
   δ[c_, d_]^2 → dim,
   δ[a1_, a2_] a3_ :→ (a3 /. a1 → a2) /; Not@FreeQ[a3, a1]};
```

In[7]:=

```
γ[d] · γ[a] · γ[c] · γ[b] · γ[d] · γ[c] · γ[a] //. rules /. CenterDot[γ[a_]] → γ[a] //
 Simplify
```

150%

guide/TestingExpressions

Wolfram Language > Expressions > Testing Expressions
Wolfram Language > Procedural Programming > Conditionals > Testing Expressions

Equality & Inequality

Equal (==), **Unequal** (!=) — numerical equality

SameQ (===), **UnsameQ** (=!=) — literal symbolic identity

Less (<) · **Greater** (>) · **LessEqual** (<=) · **GreaterEqual** (>=) · **Between**

Logical Combinations »

And (&&) · **Or** (||) · **Not** (!) · **True** · **False** · ...

AllTrue · **AnyTrue** · **NoneTrue**

Operator Forms

GreaterThan · **LessThan** · **GreaterEqualThan** · **LessEqualThan** · **EqualTo** · **UnequalTo**

Numerical Properties

NumberQ, **MachineNumberQ** — literal number, literal machine-precision number

NumericQ — numeric quantity (consisting of numbers, numeric constants, and functions)

IntegerQ · **EvenQ** · **OddQ** · **PrimeQ** · **CoprimeQ** · **AlgebraicIntegerQ** · ...

Positive · **Negative** · **NonPositive** · **NonNegative** · **PossibleZeroQ**

List-Oriented Tests

150%

Goal: Define properties of γ matrices, so that Mathematica can simplify respective expressions

Approach: We will *not* define what γ matrices are, or how they can be represented
Instead, we will only define how they behave, and what rules can be applied to expressions containing them

We will use three symbols: γ δ CenterDot ·

In[2]:=

```
SetAttributes[δ, Orderless];

NumQ[x_?NumericQ] = True;
NumQ[δ[x_, y_]] = True;
NumQ[x_] = False;

rules = {c___ · γ[a_] · γ[b_] · d___ :> c · (-γ[b] · γ[a] + 2 δ[a, b]) · d /; Not@OrderedQ[{a, b}],
   c___ · γ[a_] · γ[a_] · d___ :> c · δ[a, a] · d,
   a1___ · a2_ · a3___ :> a2 a1 · a3 /; NumQ[a2],
   a1___ · (a2_ a4_) · a3___ :> a2 a1 · a4 · a3 /; NumQ[a2],
   a1___ · (a2_ + a4_) · a3___ :> a1 · a2 · a3 + a1 · a4 · a3,
   CenterDot[a__] :> Flatten[CenterDot[a]],
   δ[c_, c_] → dim,
   δ[c_, d_]^2 → dim,
   δ[a1_, a2_] a3_ :> (a3 /. a1 → a2) /; Not@FreeQ[a3, a1]};
```

In[7]:=

```
γ[d] · γ[a] · γ[c] · γ[b] · γ[d] · γ[c] · γ[a] //. rules /. CenterDot[γ[a_]] → γ[a] //
 Simplify
```

150%

Goal: Define properties of γ matrices, so that Mathematica can simplify respective expressions

Approach: We will *not* define what γ matrices are, or how they can be represented
Instead, we will only define how they behave, and what rules can be applied to expressions containing them

We will use three symbols: γ δ CenterDot ·

```
SetAttributes[δ, Orderless];

NumQ[x_?NumericQ] = True;
NumQ[δ[x_, y_]] = True;
NumQ[x_] = False;

rules = {c___ · γ[a_] · γ[b_] · d___ :→ c · (-γ[b] · γ[a] + 2 δ[a, b]) · d /; Not@OrderedQ[{a, b}],
   CenterDot[c___, γ[a_], γ[a_], d___] :→ c · δ[a, a] · d,
   a1___ · a2_ · a3___ :→ a2 a1 · a3 /; NumQ[a2],
   a1___ · (a2_ a4_) · a3___ :→ a2 a1 · a4 · a3 /; NumQ[a2],
   a1___ · (a2_ + a4_) · a3___ :→ a1 · a2 · a3 + a1 · a4 · a3,
   CenterDot[a__] :→ Flatten[CenterDot[a]],
   δ[c_, c_] → dim,
   δ[c_, d_]^2 → dim,
   δ[a1_, a2_] a3_ :→ (a3 /. a1 → a2) /; Not@FreeQ[a3, a1]};
```

In[7]:=
```
γ[d] · γ[a] · γ[c] · γ[b] · γ[d] · γ[c] · γ[a] //. rules /. CenterDot[γ[a_]] → γ[a] //
 Simplify
```

150%

$$\delta_{ij} \, T_{jk} \rightarrow T_{ik}$$

$$\delta_{ij} \, T_{k\ell} \not\Rightarrow T_{k\ell}$$

Goal: Define properties of γ matrices, so that Mathematica can simplify respective expressions

Approach: We will *not* define what γ matrices are, or how they can be represented
Instead, we will only define how they behave, and what rules can be applied to expressions containing them

We will use three symbols: γ δ CenterDot ·

```
SetAttributes[δ, Orderless];

NumQ[x_?NumericQ] = True;
NumQ[δ[x_, y_]] = True;
NumQ[x_] = False;

rules = {c___ · γ[a_] · γ[b_] · d___ ⧴ c · (-γ[b] · γ[a] + 2 δ[a, b]) · d /; Not@OrderedQ[{a, b}],
   CenterDot[c___, γ[a_], γ[a_], d___] ⧴ c · δ[a, a] · d,
   a1___ · a2_ · a3___ ⧴ a2 a1 · a3 /; NumQ[a2],
   a1___ · (a2_ a4_) · a3___ ⧴ a2 a1 · a4 · a3 /; NumQ[a2],
   a1___ · (a2_ + a4_) · a3___ ⧴ a1 · a2 · a3 + a1 · a4 · a3,
   CenterDot[a__] ⧴ Flatten[CenterDot[a]],
   δ[c_, c_] → dim,
   δ[c_, d_]^2 → dim,
   δ[i_, j_] a_ ⧴ (a /. i → j) /; Not@FreeQ[a, i]};
```

In[7]:=

```
γ[d] · γ[a] · γ[c] · γ[b] · γ[d] · γ[c] · γ[a] //. rules /. CenterDot[γ[a_]] → γ[a] //
 Simplify
```

150%

```
NumQ[x_?NumericQ] = True;
NumQ[δ[x_, y_]] = True;
NumQ[x_] = False;

rules = {c___ · γ[a_] · γ[b_] · d___ :→ c · (-γ[b] · γ[a] + 2 δ[a, b]) · d /; Not@OrderedQ[{a, b}],
   CenterDot[c___, γ[a_], γ[a_], d___] :→ c · δ[a, a] · d,
   a1___ · a2_ · a3___ :→ a2 a1 · a3 /; NumQ[a2],
   a1___ · (a2_ a4_) · a3___ :→ a2 a1 · a4 · a3 /; NumQ[a2],
   a1___ · (a2_ + a4_) · a3___ :→ a1 · a2 · a3 + a1 · a4 · a3,
   CenterDot[a__] :→ Flatten[CenterDot[a]],
   CenterDot[γ[a_]] :> γ[a],
   δ[c_, c_] → dim,
   δ[c_, d_]^2 → dim,
   δ[i_, j_] a_ :→ (a /. i → j) /; Not@FreeQ[a, i]};
```

In[24]:=
```
γ[d] · γ[a] · γ[c] · γ[b] · γ[d] · γ[c] · γ[a] //. rules (*/.CenterDot[γ[a_]]→γ[a]*) //
 Simplify
```

Out[24]=
$$-\left(-16 + 24\,\mathrm{dim} - 10\,\mathrm{dim}^2 + \mathrm{dim}^3\right)\,\mathrm{CenterDot}[\gamma[b]]$$

Deck of cards

```
NumQ[x_?NumericQ] = True;
NumQ[δ[x_, y_]] = True;
NumQ[x_] = False;

rules = {c___ · γ[a_] · γ[b_] · d___ :> c · (-γ[b] · γ[a] + 2 δ[a, b]) · d /; Not@OrderedQ[{a, b}],
   CenterDot[c___, γ[a_], γ[a_], d___] :> c · δ[a, a] · d,
   a1___ · a2_ · a3___ :> a2 a1 · a3 /; NumQ[a2],
   a1___ · (a2_ a4_) · a3___ :> a2 a1 · a4 · a3 /; NumQ[a2],
   a1___ · (a2_ + a4_) · a3___ :> a1 · a2 · a3 + a1 · a4 · a3,
   CenterDot[a__] :> Flatten[CenterDot[a]],
   CenterDot[γ[a_]] :> γ[a],
   a1_ CenterDot[γ[a_]] :> γ[a] /; NumQ[a1],
   δ[c_, c_] → dim,
   δ[c_, d_]^2 → dim,
   δ[i_, j_] a_ :> (a /. i → j) /; Not@FreeQ[a, i]};
```

In[42]:=

```
γ[d] · γ[a] · γ[c] · γ[b] · γ[d] · γ[c] · γ[a] //. rules (*/.CenterDot[γ[a_]]→γ[a]*) //
 Simplify
```

Out[42]=

$(8 + 6\,\mathrm{dim} - 5\,\mathrm{dim}^2 + \mathrm{dim}^3)\,\gamma[b]$

Deck of cards

150%

```
NumQ[x_?NumericQ] = True;
NumQ[δ[x_, y_]] = True;
NumQ[x_] = False;

rules = {c___ · γ[a_] · γ[b_] · d___ :→ c · (-γ[b] · γ[a] + 2 δ[a, b]) · d /; Not@OrderedQ[{a, b}],
   CenterDot[c___, γ[a_], γ[a_], d___] :→ c · δ[a, a] · d,
   a1___ · a2_ · a3___ :→ a2 a1 · a3 /; NumQ[a2],
   a1___ · (a2_ a4_) · a3___ :→ a2 a1 · a4 · a3 /; NumQ[a2],
   a1___ · (a2_ + a4_) · a3___ :→ a1 · a2 · a3 + a1 · a4 · a3,
   CenterDot[a__] :→ Flatten[CenterDot[a]],
   CenterDot[γ[a_]] :→ γ[a],
   a1_ CenterDot[γ[a_]] :→ γ[a] /; NumQ[a1],
   δ[c_, c_] → dim,
   δ[c_, d_]^2 → dim,
   δ[i_, j_] a_ :→ (a /. i → j) /; Not@FreeQ[a, i]};
```

In[42]:=

```
γ[d] · γ[a] · γ[c] · γ[b] · γ[d] · γ[c] · γ[a] //. rules (*/.CenterDot[γ[a_]]→γ[a]*) //
 Simplify
```

Out[42]=

$$\left(8 + 6\,\mathrm{dim} - 5\,\mathrm{dim}^2 + \mathrm{dim}^3\right)\gamma[\mathrm{b}]$$

Deck of cards

150%

```
NumQ[x_?NumericQ] = True;
NumQ[δ[x_, y_]] = True;
NumQ[x_] = False;

rules = {c___ · γ[a_] · γ[b_] · d___ :> c · (-γ[b] · γ[a] + 2 δ[a, b]) · d /; Not@OrderedQ[{a, b}],
    CenterDot[c___, γ[a_], γ[a_], d___] :> c · δ[a, a] · d,
    a1___ · a2_ · a3___ :> a2 a1 · a3 /; NumQ[a2],
    a1___ · (a2_ a4_) · a3___ :> a2 a1 · a4 · a3 /; NumQ[a2],
    a1___ · (a2_ + a4_) · a3___ :> a1 · a2 · a3 + a1 · a4 · a3,
    CenterDot[a__] :> Flatten[CenterDot[a]],
    δ[c_, c_] → dim,
    δ[c_, d_]^2 → dim,
    δ[i_, j_] a_ :> (a /. i → j) /; Not@FreeQ[a, i]};
```

In[49]:=
```
γ[d] · γ[a] · γ[c] · γ[b] · γ[d] · γ[c] · γ[a] //. rules /. CenterDot[γ[a_]] → γ[a] //
 Simplify
```

Out[49]= $-\left(-16 + 24\,\mathrm{dim} - 10\,\mathrm{dim}^2 + \mathrm{dim}^3\right)\gamma[\mathrm{b}]$

Deck of cards

```
Hearts = Style[♡, {Red, Large}];
```

150%

```
NumQ[x_?NumericQ] = True;
NumQ[δ[x_, y_]] = True;
NumQ[x_] = False;

rules = {c___ · γ[a_] · γ[b_] · d___ :> c · (-γ[b] · γ[a] + 2 δ[a, b]) · d /; Not@OrderedQ[{a, b}],
   CenterDot[c___, γ[a_], γ[a_], d___] :> c · δ[a, a] · d,
   a1___ · a2_ · a3___ :> a2 a1 · a3 /; NumQ[a2],
   a1___ · (a2_ a4_) · a3___ :> a2 a1 · a4 · a3 /; NumQ[a2],
   a1___ · (a2_ + a4_) · a3___ :> a1 · a2 · a3 + a1 · a4 · a3,
   CenterDot[a__] :> Flatten[CenterDot[a]],
   δ[c_, c_] → dim,
   δ[c_, d_]^2 → dim,
   δ[i_, j_] a_ :> (a /. i → j) /; Not@FreeQ[a, i]};
```

In[49]:=

```
γ[d] · γ[a] · γ[c] · γ[b] · γ[d] · γ[c] · γ[a] //. rules /. CenterDot[γ[a_]] → γ[a] //
 Simplify
```

Out[49]= $-\left(-16 + 24\,\mathrm{dim} - 10\,\mathrm{dim}^2 + \mathrm{dim}^3\right)\,\gamma[\mathrm{b}]$

1. We don't define what γ δ or CenterDot actually are, or how they are represented
2. We only define rules that modify expressions containing these
3. The set of rules needs to be "constructive", i.e. it needs to terminate when applied repeatedly

Deck of cards

150%

Computational Physics

Erik Schnetter
Perimeter Institute, September 2, 2016

Computational Physics

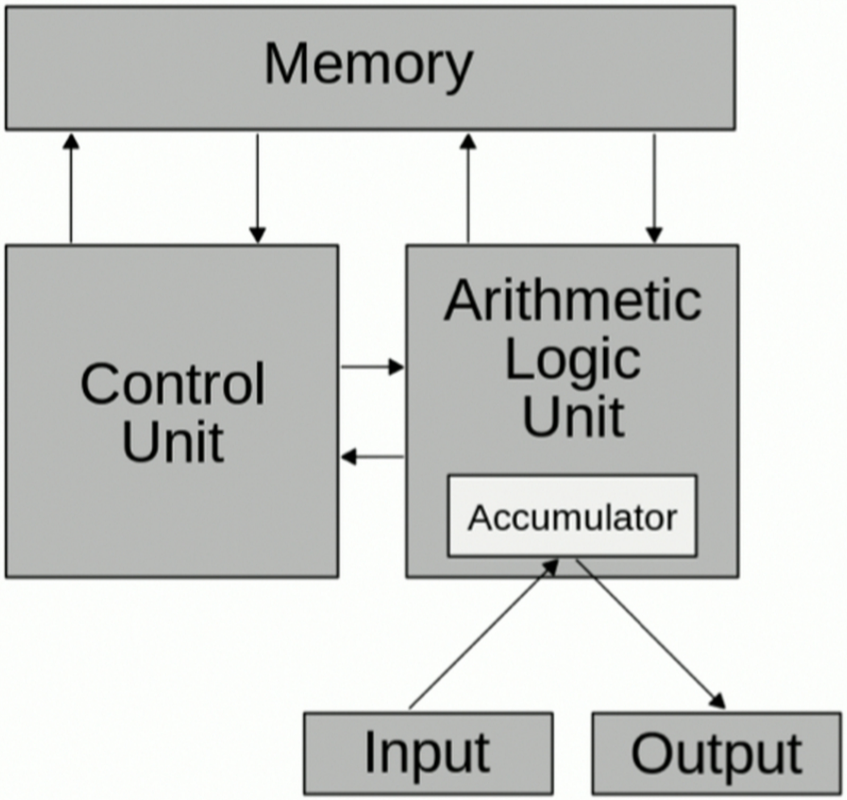
- High-level introduction to computational methods

1. Computing architecture (“what is a computer and how does it work”)
2. Numerical analysis (“how do I solve a PDE on a computer”)

Algorithm Development

- Naïve idea:
  - Von Neumann (and others) developed a theoretical model of computing
  - Take an algorithm ("constructive proof"), map it to this model, implement it in a computer language, done
- In practice:
  - What is possible (and what is efficient) is not determined by an abstract model, but by current-day hardware technology
  - "Good" algorithms today look very different than 20 years ago
  - (… not because people today are more clever!)

Von Neumann Architecture



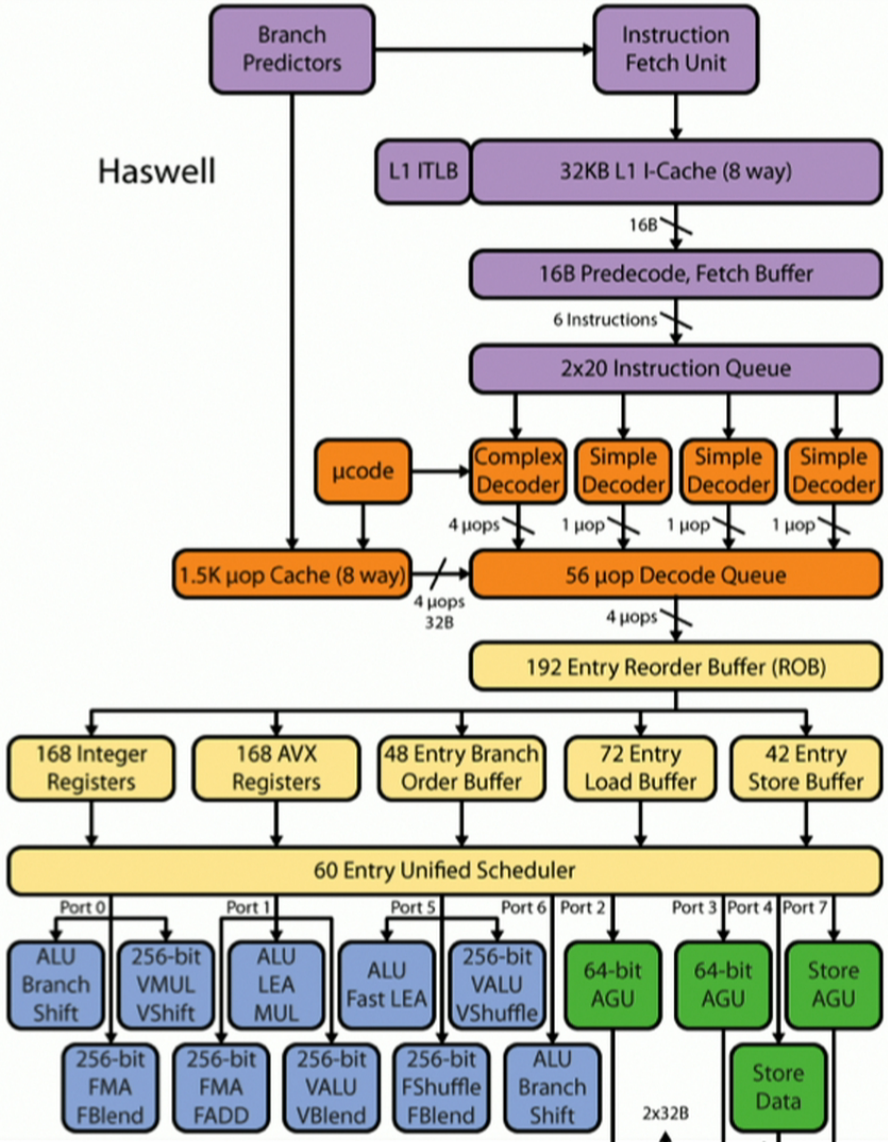
Sequential Execution:

x := a + b

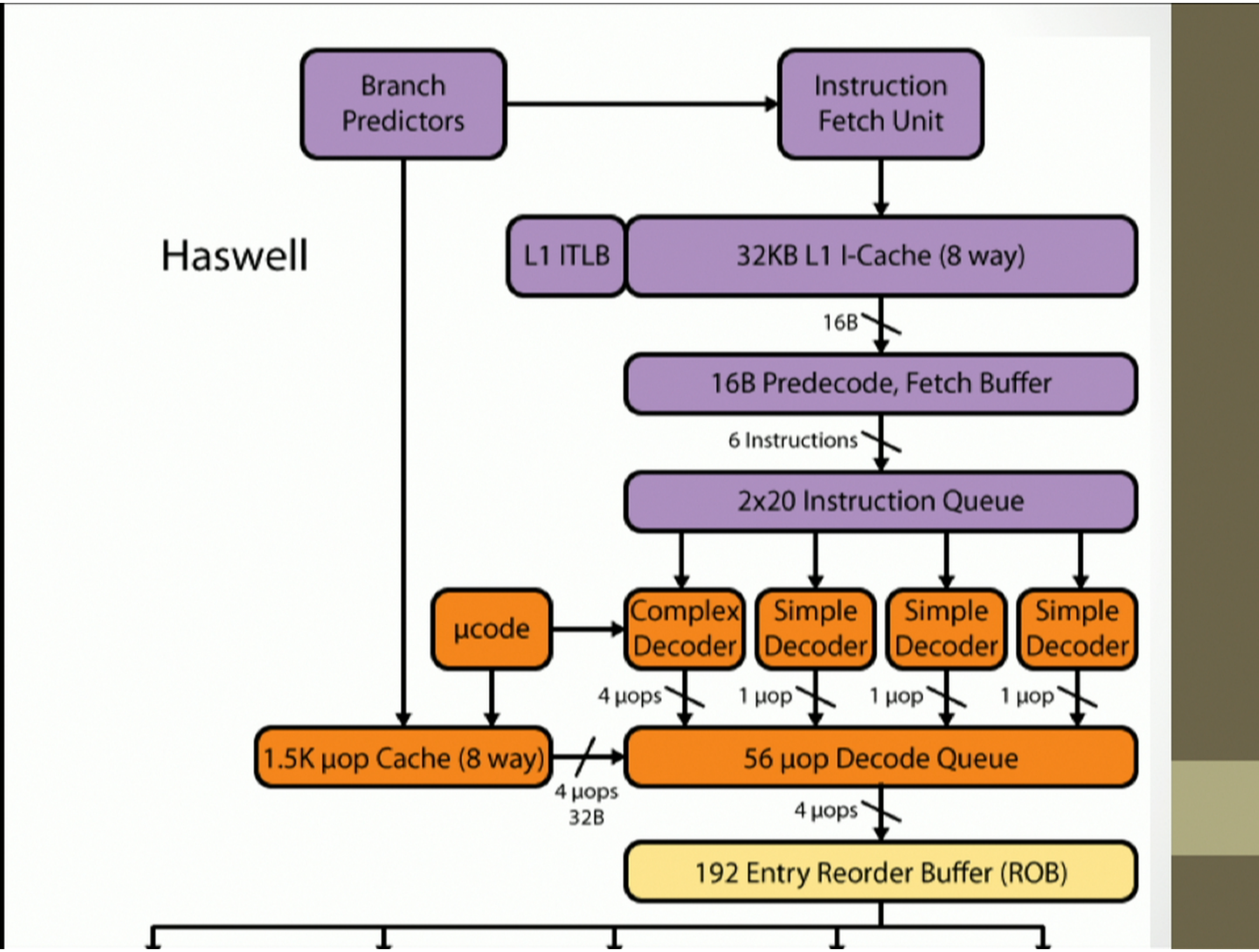
1. read instruction
2. read a
3. read b
4. add
5. write x

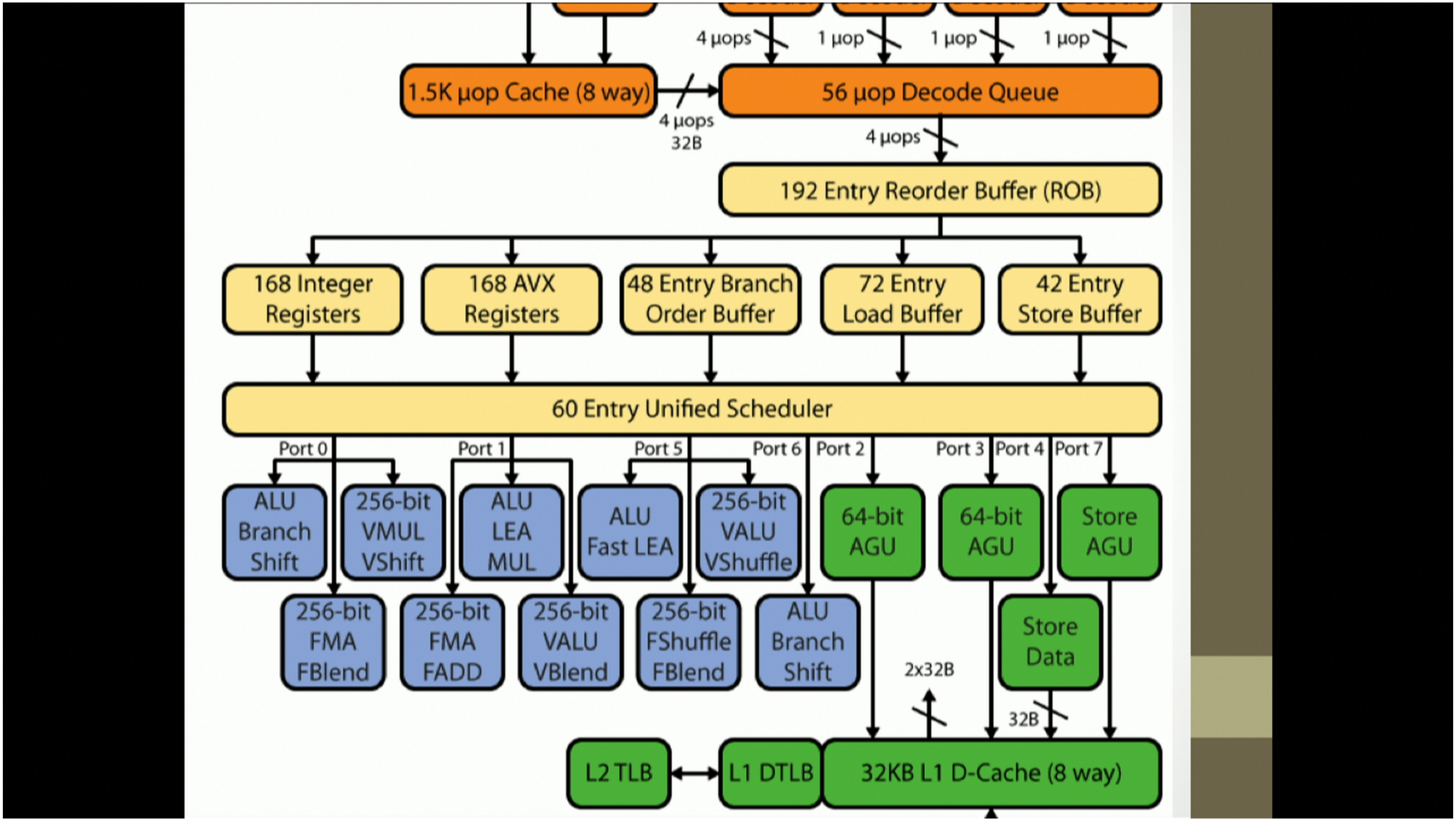
[Wikipedia]

Modern Intel processor



[IDF 2012]

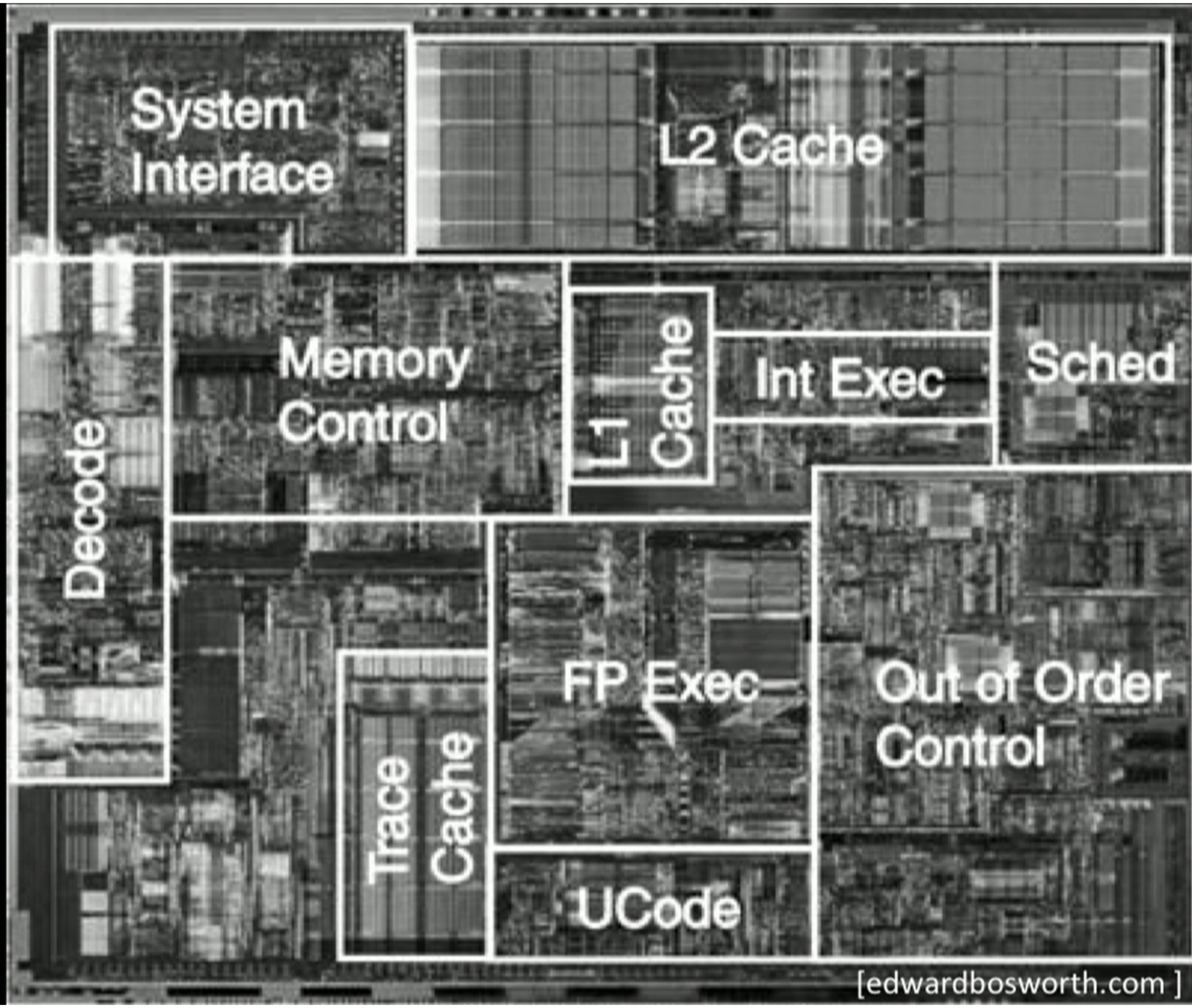

Haswell
Branch Predictors
Instruction Fetch Unit
L1 ITLB
32KB L1 I-Cache (8 way)
16B
16B Predecode, Fetch Buffer
6 Instructions
2x20 Instruction Queue
µcode
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
4 µops
1 µop
1 µop
1 µop
1.5K µop Cache (8 way)
4 µops
32B
56 µop Decode Queue
4 µops
192 Entry Reorder Buffer (ROB)

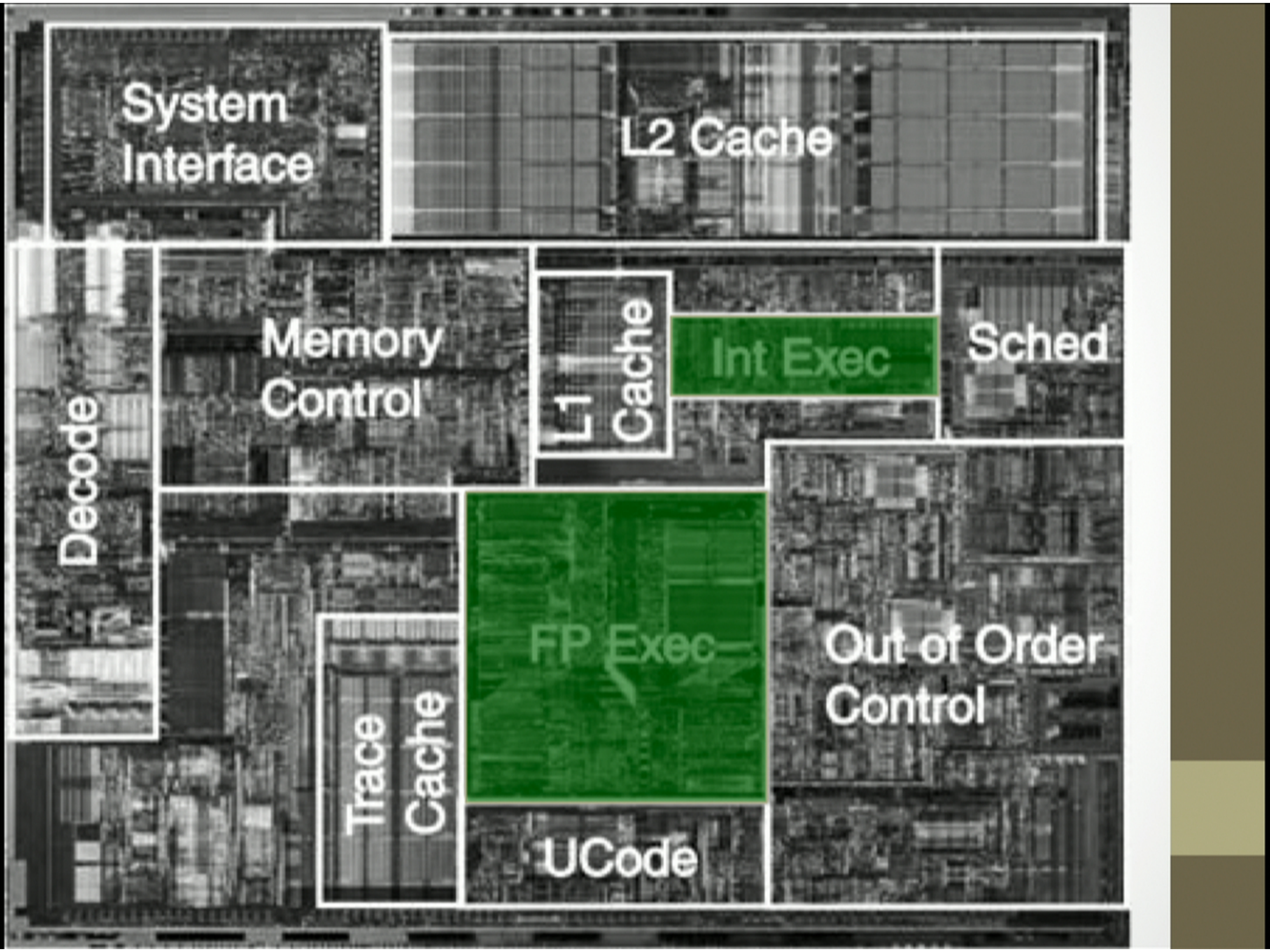

4 µops
1 µop
1 µop
1 µop
1.5K µop Cache (8 way)
56 µop Decode Queue
4 µops
32B
4 µops
192 Entry Reorder Buffer (ROB)
168 Integer Registers
168 AVX Registers
48 Entry Branch Order Buffer
72 Entry Load Buffer
42 Entry Store Buffer
60 Entry Unified Scheduler
Port 0
Port 1
Port 5
Port 6
Port 2
Port 3
Port 4
Port 7
ALU Branch Shift
256-bit VMUL VShift
ALU LEA MUL
ALU Fast LEA
256-bit VALU VShuffle
64-bit AGU
64-bit AGU
Store AGU
256-bit FMA FBlend
256-bit FMA FADD
256-bit VALU VBlend
256-bit FShuffle FBlend
ALU Branch Shift
Store Data
2x32B
32B
L2 TLB
L1 DTLB
32KB L1 D-Cache (8 way)

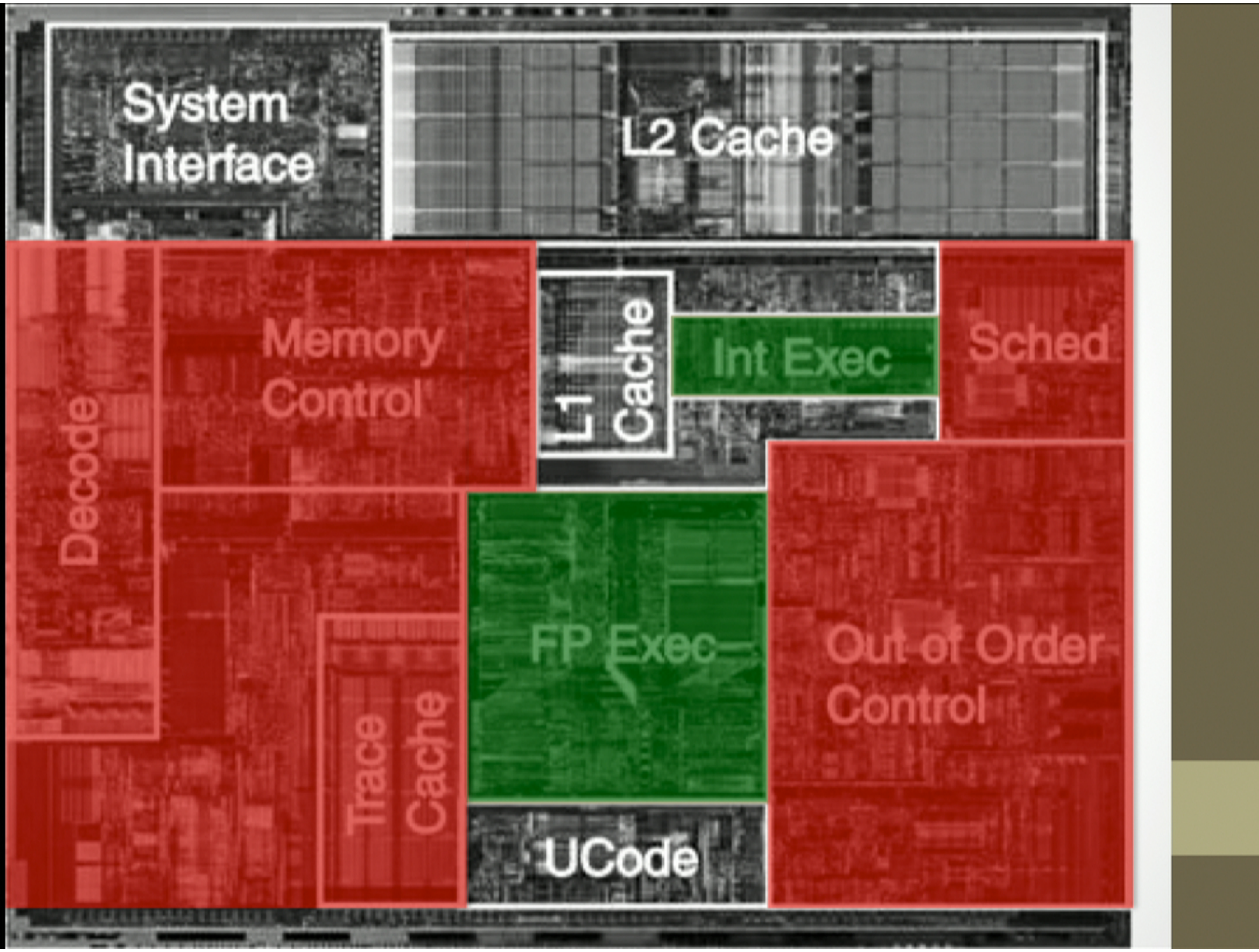
Sequential vs. Parallel

- Modern CPUs are highly parallel
  - Many cores (28)
  - Each core is superscalar (10…100)
  - Vector ("SIMD") operations (4)
  - FMA (a*b+c) (2)

- Algorithms need to be parallel (many independent steps), not sequential (each step depending on the previous)
  - Bad: Update a vector element-by-element
  - Good: Create a new vector from an existing one, each element independently


System
Interface
L2 Cache
Memory
Control
L1
Cache
Int Exec
Sched
Decode
FP Exec
Out of Order
Control
Trace
Cache
UCode
[edwardbosworth.com]


System Interface
L2 Cache
Memory Control
L1 Cache
Int Exec
Sched
Decode
FP Exec
Out of Order Control
Trace Cache
UCode


System Interface
L2 Cache
Memory Control
L1 Cache
Int Exec
Sched
Decode
FP Exec
Out of Order Control
Trace Cache
UCode

Performance Measures

- *Theoretical Peak Performance*
  1. Computing: Floating point operations per second (Flop/sec)
  2. Data access: Bytes per second (Byte/sec)

- Typical numbers for a modern workstation:
  - 1,000 GFlop/sec (Giga, 10^9)
  - 100 GByte/sec
  - Ideal ratio: 0.1 Byte/Flop
  - Double precision add: 24 Byte/Flop

- Memory (RAM): 100 GByte

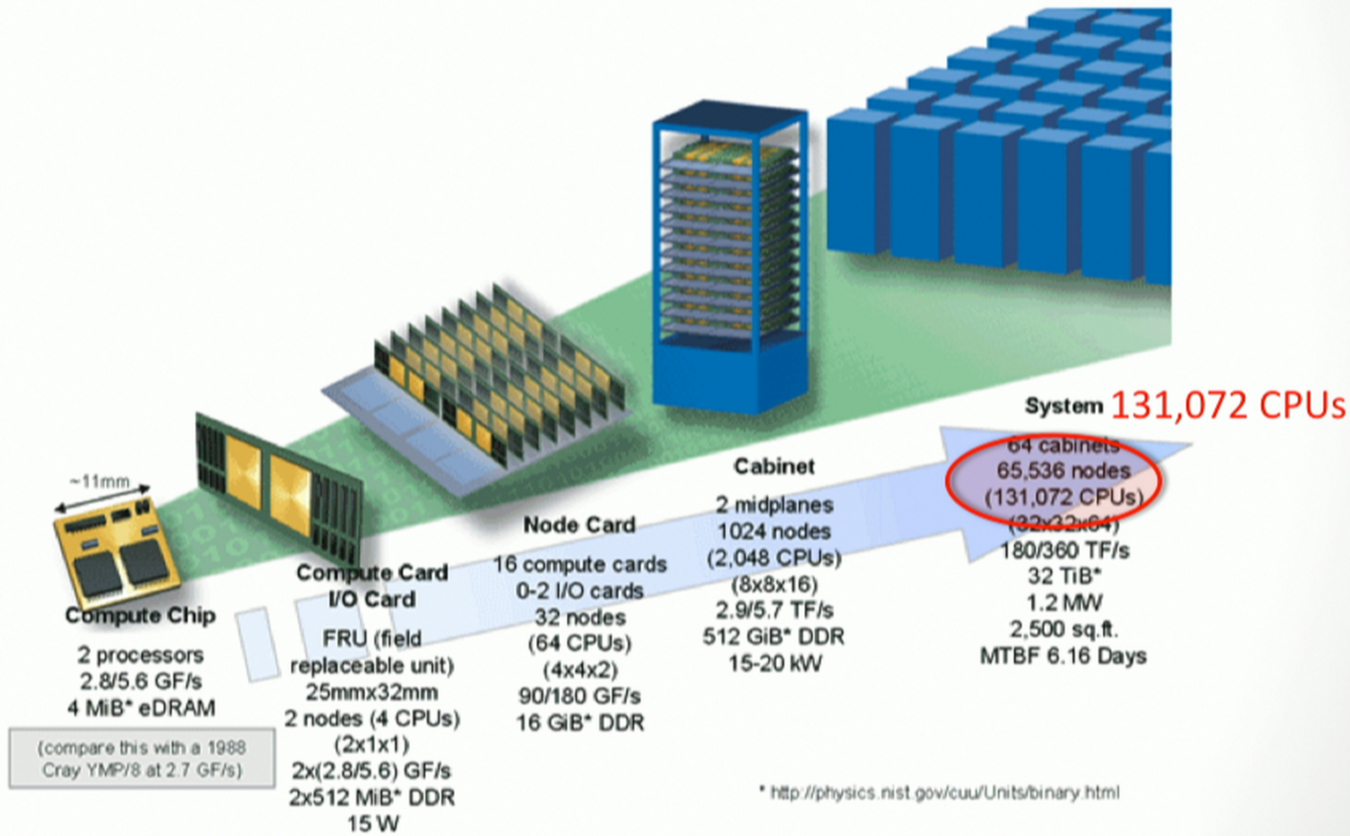
Algorithm characteristics

- Algorithms can be characterized by the number of calculations they perform (Flop) and amount of data they process (Byte)

- Matrix multiplication (n*n matrices, 8 Byte per number):
  - $2n^3$ Flop
  - $24n^2$ Byte
  - 12/n Byte/Flop
  - Thus: compute bound even for moderate n

- Matrix addition:
  - n^2 Flop
  - $24n^2$ Byte
  - Ratio: 24 Byte/Flop
  - Thus: memory bound

Supercomputer architecture

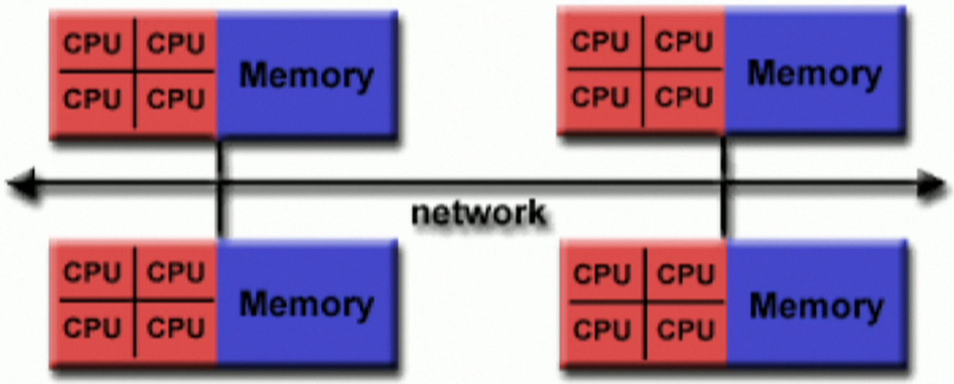


Supercomputer?

Top500

[top500.org]

Distributed Memory



[computing.llnl.gov]

- Need to *partition* data structures
- Need to *exchange information* at (artificial) inter-node boundaries
- Permeates the whole program, quite difficult in practice

Software Architecture

- Every decade or two, there is a paradigm shift in software architecture
  - 1980s: modular programming (“information hiding”)
  - 1990s: object oriented programming (generic programming)
  - 2000s: parallel programming (multi-threading)
  - 2010s: (?) functional (declarative) programming

Parallel Programming Algebra

- Problems: Access conflicts, deadlocks
  - Naively parallel programs do not form a group

- Relatively recent solution, found today e.g. in C++, Python, Julia:
  - *Future*: “black box” holding the result of a calculation that is not yet finished
  - Accessing a future automatically waits until the result is ready
  - *Async*: (asynchronous execution): “function call” returning a future, without waiting for the result
  - C++: Future is a “container” (or could be)

Functional Programming

- Dichotomy between mathematics and programming:
  1. Math is about eternal truths (there is no “time” in a proof)
  2. Programs execute sequentially
- How can one prove statements about programs?

- Functional programming:
  - Design programs to be order-independent (as much as possible)
  - Remove distinction between data and functions

```
let x = 1
  sin(x)
end
```

```
let f = sin
   f(1)
end
```

julia

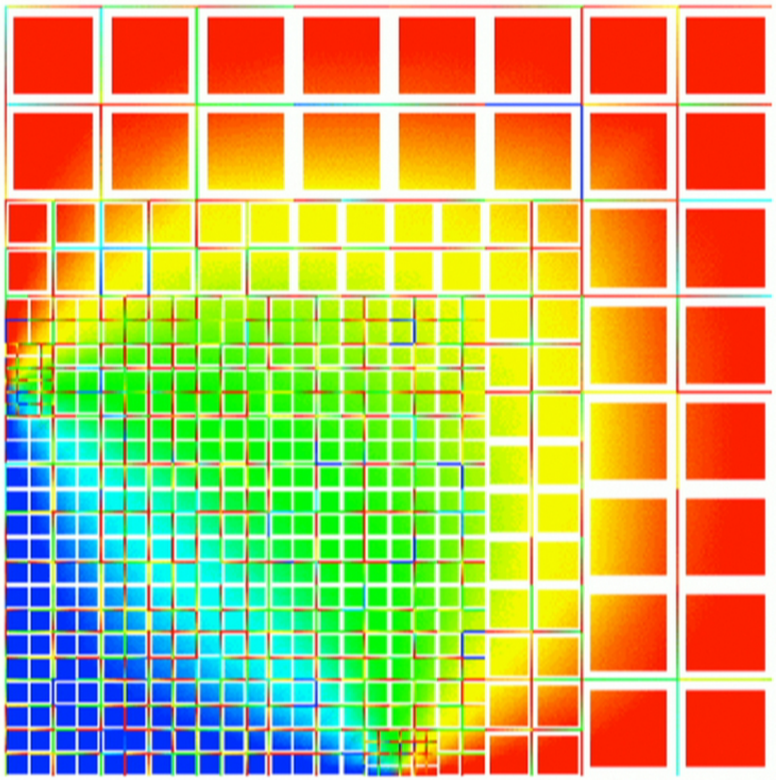
- From juliacode.org:
  - high-level, high-performance dynamic programming language for technical computing
  - sophisticated compiler
  - distributed parallel execution
  - numerical accuracy
  - extensive mathematical function library
  - mature, best-of-breed open source C and Fortran libraries for linear algebra, random number generation, …
  - powerful browser-based graphical notebook interface
- Note: only few years old, still immature

- From juliacode.org:
  - high-level, high-performance dynamic programming language for technical computing
  - sophisticated compiler
  - distributed parallel execution
  - numerical accuracy
  - extensive mathematical function library
  - mature, best-of-breed open source C and Fortran libraries for linear algebra, random number generation, …
  - powerful browser-based graphical notebook interface
- Note: only few years old, still immature

Numerical Analysis

- How to solve a PDE on a computer
  - How to approximate fields via finite-dimensional spaces: discretization

Discretization



[wissrech.iam.uni-bonn.de]

Common Discretization Methods

- Finite Differences:
  - Sample solution at grid points, interpolate to define continuum
- Finite Elements:
  - Define elements with vertices, edges, faces, …; often used with differential forms
- Finite Volumes:
  - Split domain into cells; define fluxes through faces; commonly used for conservation laws
- Particle methods (*Lagrangian methods*):
  - Split matter/charges into "particles" that move and interact with neighbours; often used for conserved quantities with irregular distributions

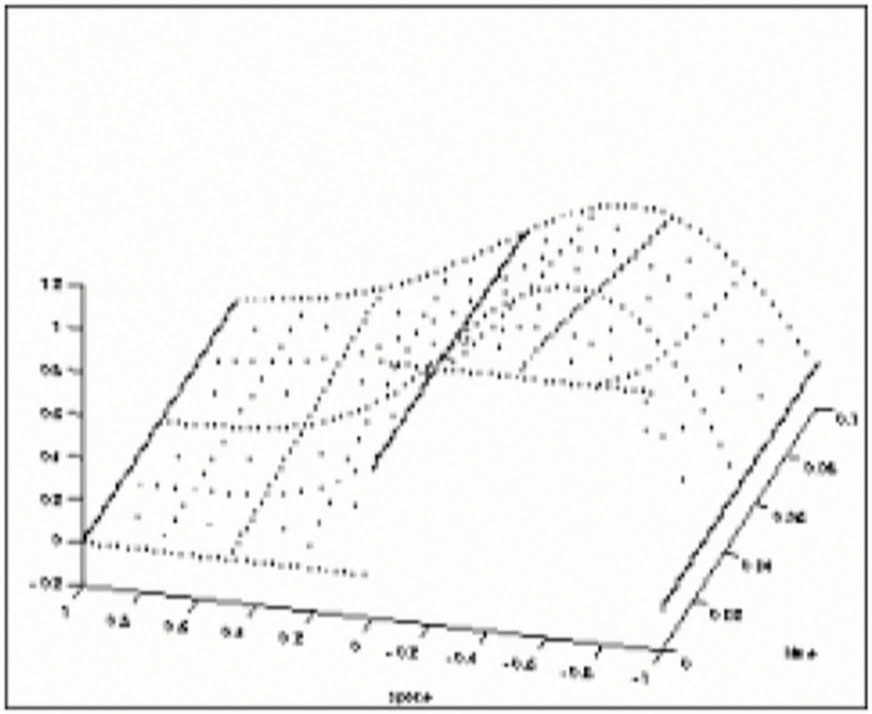
Mimetic Discretizations

- Some continuum properties may/will not hold in the discrete, e.g.:
  - Conservation
  - Integration by parts
  - Constraints
  - Sharp discontinuities
- Typically: Need to choose which properties are important for a system, represent these faithfully ("mimetic"), drop others
  - E.g. integration by parts: can be recovered by careful choice of derivative operators, integration weights, boundary definition

Sparse grids

Can represent
D-dimensional domain
with $O(2^D N)$ coefficients

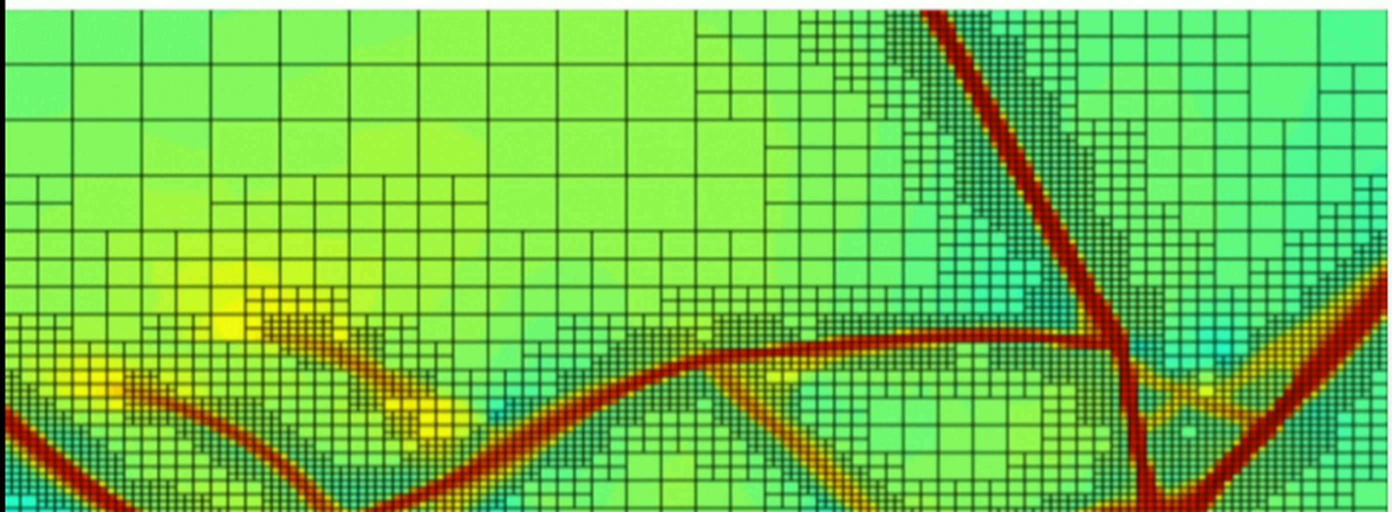
Adapt grid to PDE



[wissrech.iam.uni-bonn.de]

Adaptive Mesh Refinement

Dynamically adapt mesh to solution U



[jupiter.ethz.ch]

Summary

- Efficiency of algorithms depends on available computing architecture
  - Parallelism (independent operations, not sequential)
  - Data locality (operate on "neighbouring" data)
- Large problems require distributed memory
  - Partition data structure
- Modern software engineering:
  - Powerful concepts from group theory, category theory