Domain Specific Languages past, present and future

Mary Sheeran

Emil Axelsson

Chalmers

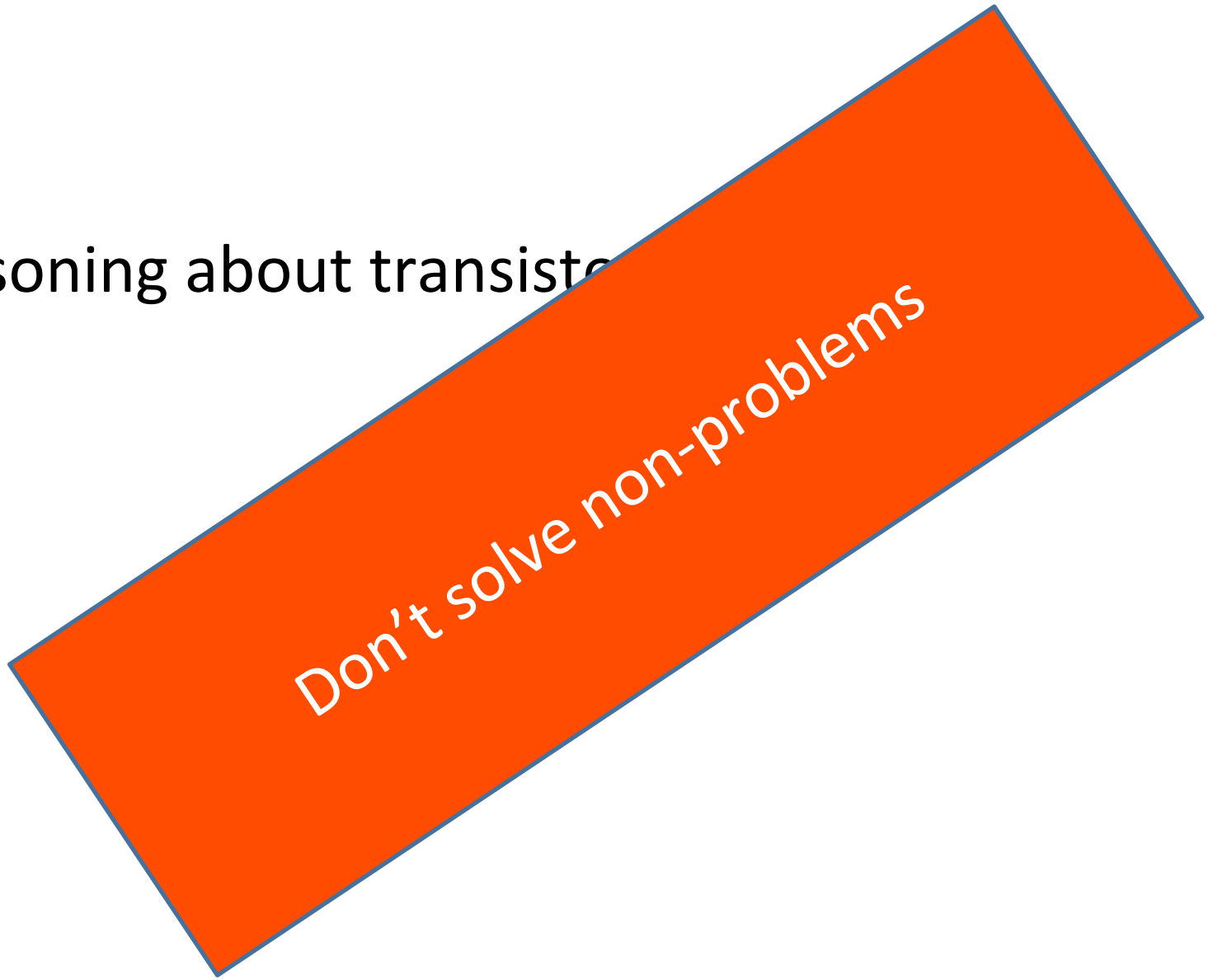

HAVFRUEN

Concurrency

Protocols

Reasoning about transistor circuits (in CSP)

Reasoning about transiste



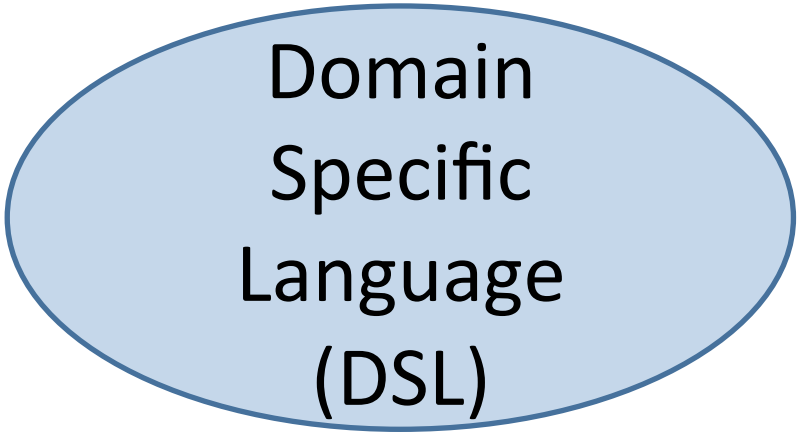
programming languages

programming languages

Backus FP -> μFP

programming languages

Backus FP -> μFP



Domain-specific language (DSL)

From [Kosar et al, 2008]:

"provide a notation tailored toward an application domain"

"based only on the relevant concepts and features of that domain"

"a means of describing and generating members of a program family within a given domain, without the need for knowledge about general programming"

"offers substantial gains in productivity"

thanks to E. Axelsson

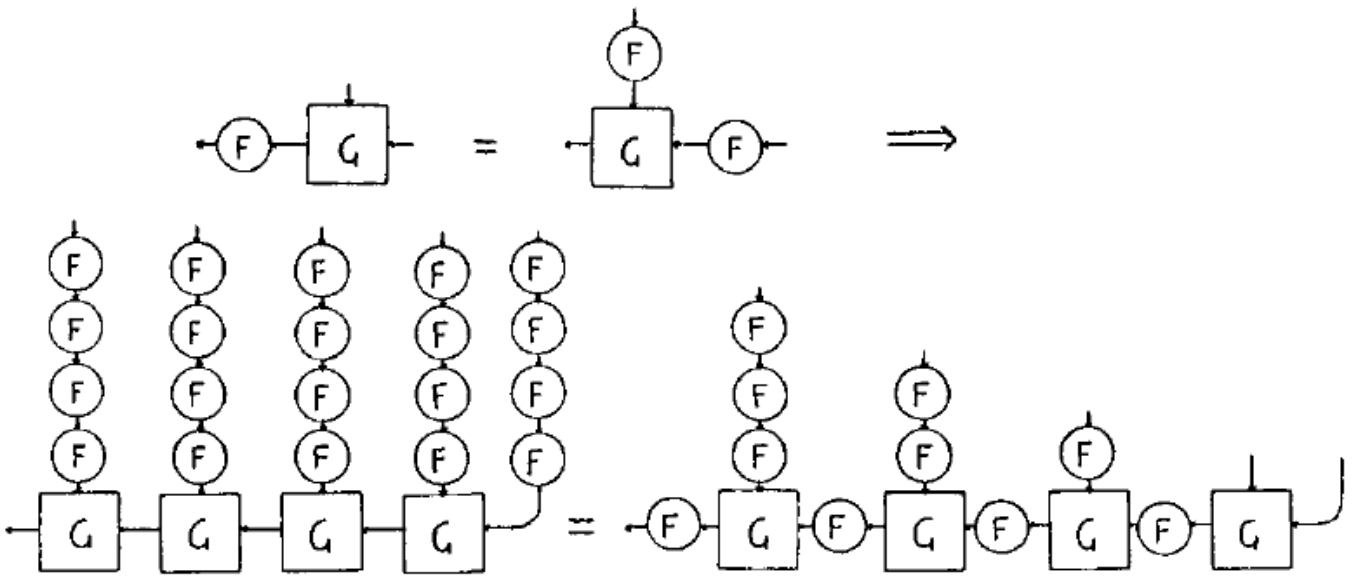
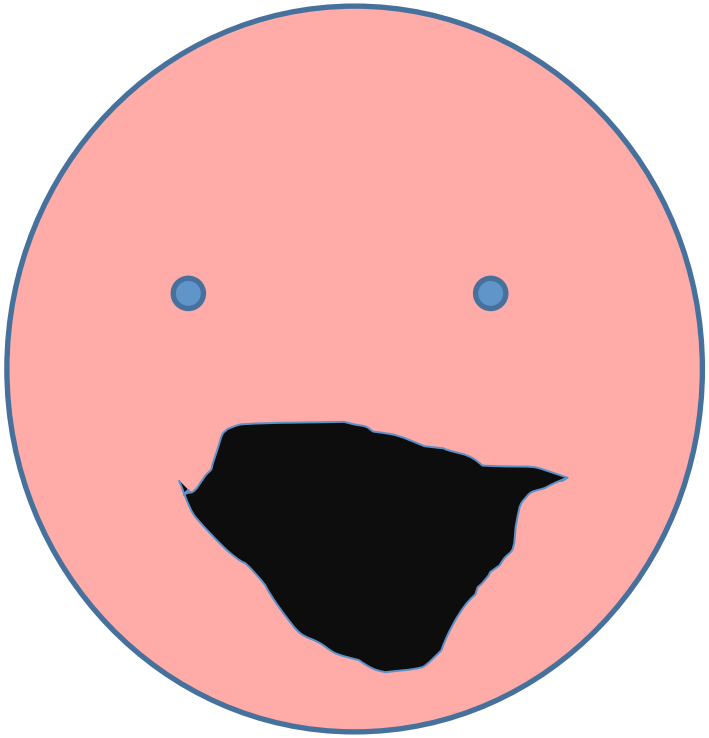
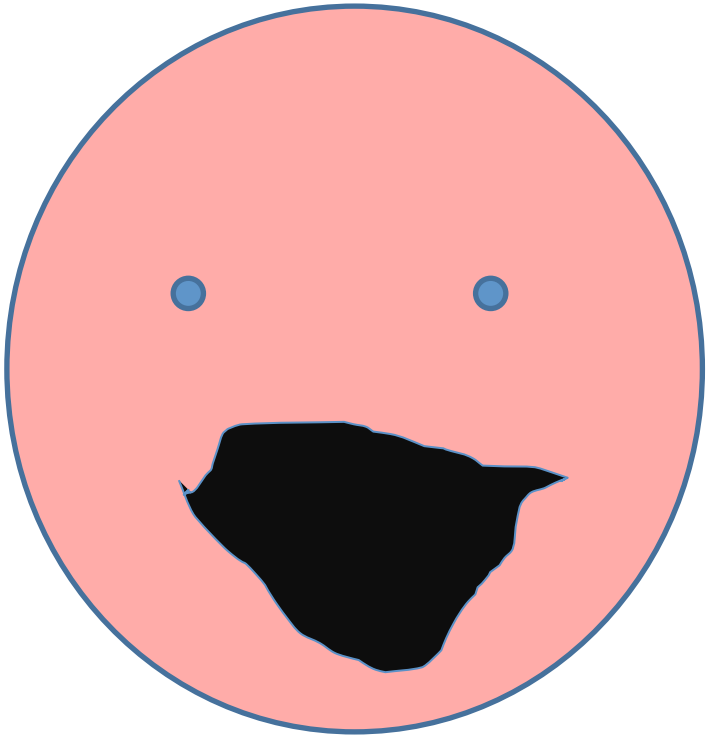


FIG 8. F . G = G . @F => /G . <>F = /(F.G) . mL>F (/>)

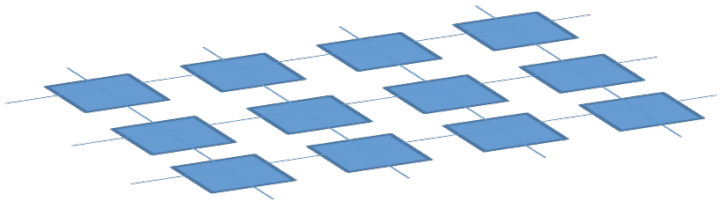




HOW DARE YOU TELL MY DESIGNERS WHAT TO DO!

Users!

Plessey



Users!

Plessey

Plessey designers write

Using muFP, the array processing element was described in just one line of code and the complete array required four lines of muFP description. muFP enabled the effects of adding or moving data latches within the array to be assessed quickly. Since the results were in symbolic form it was clear where and when data within the results was input into the array making it simple to examine the data-flow within the array and change it as desired. This was found to be a very useful way to learn about the data dependencies within the array.

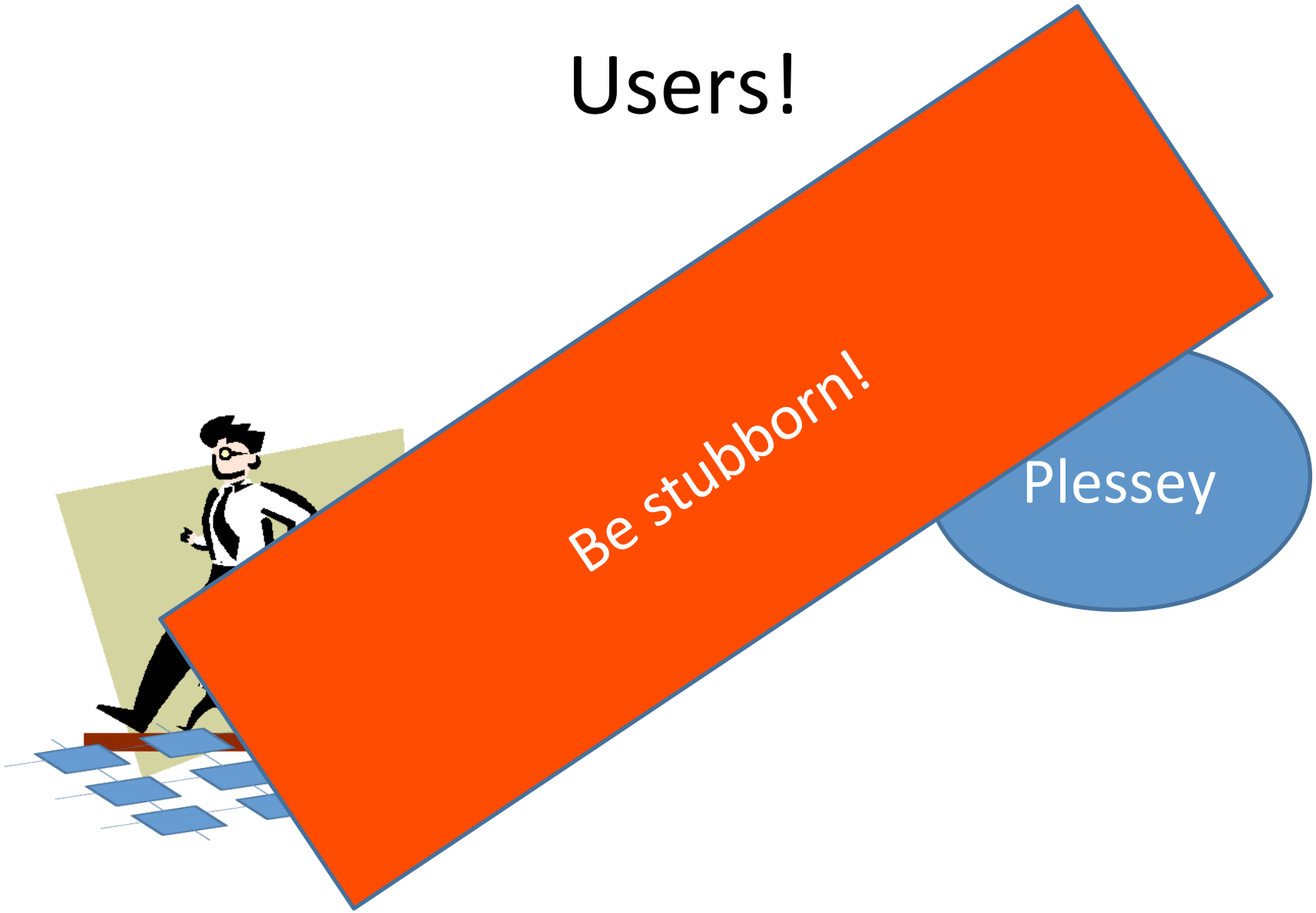
[...]

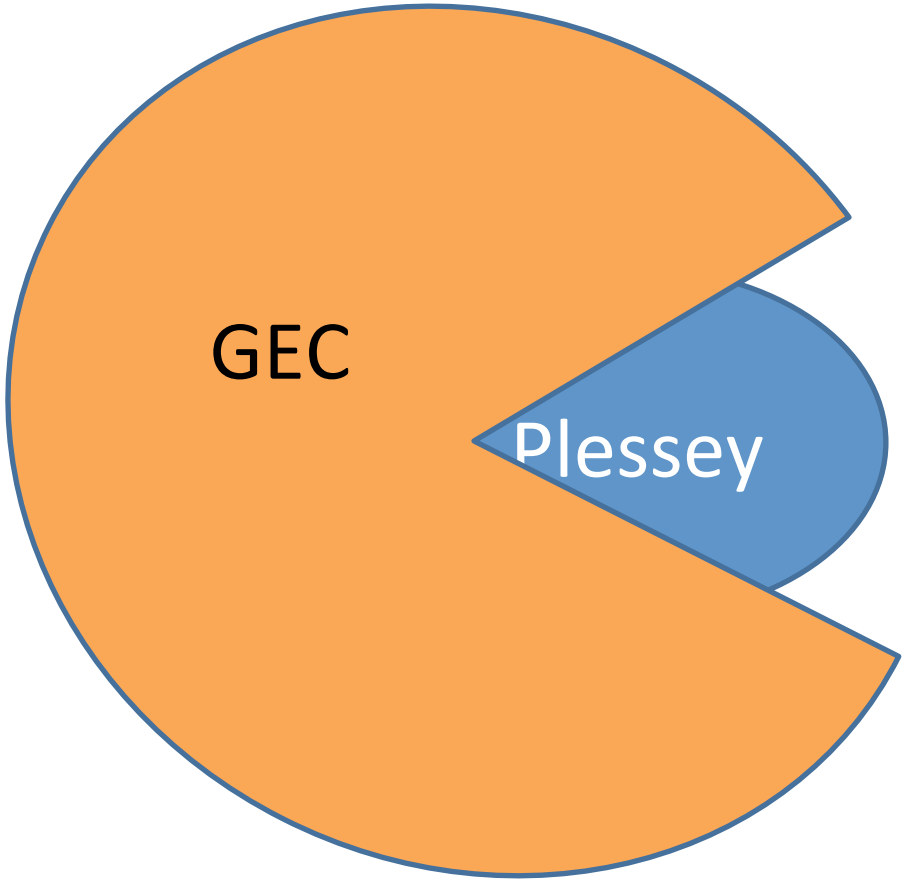
From the experience gained on the design, the most important consideration when designing array processors is to ensure that the processor input/output requirements can be met easily and without sacrificing array performance. The most difficult part of the design task is not the design of the computation units but the design of the data paths and associated storage devices. It is essential to have the right design tools to aid and improve the design process. Early use of tools to explore the flow of data within and around the array and to understand the data of requirements of the array is important. muFP has been shown to be useful for this purpose.

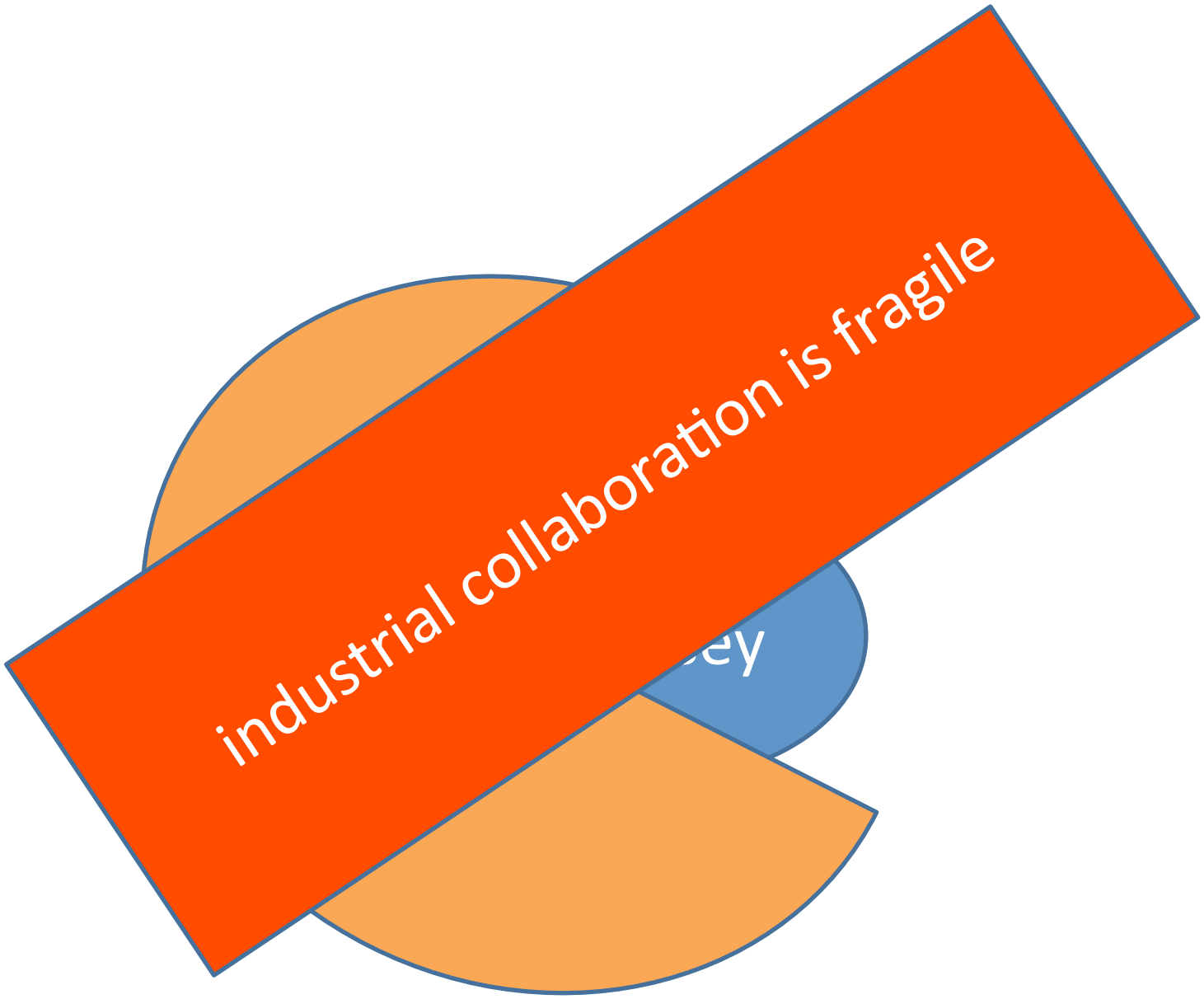
Bhandal et al, An array processor for video picture motion estimation, Systolic Array Processors, 1990, Prentice Hall

work with Plessey done by G. Jones and W. Luk

Users!




GEC
Plessey


industrial collaboration is fragile

Ruby (relational version of muFP)

Lava (muFP in Haskell, waaaay better)

SAT-based verification (distracted by a clever man)

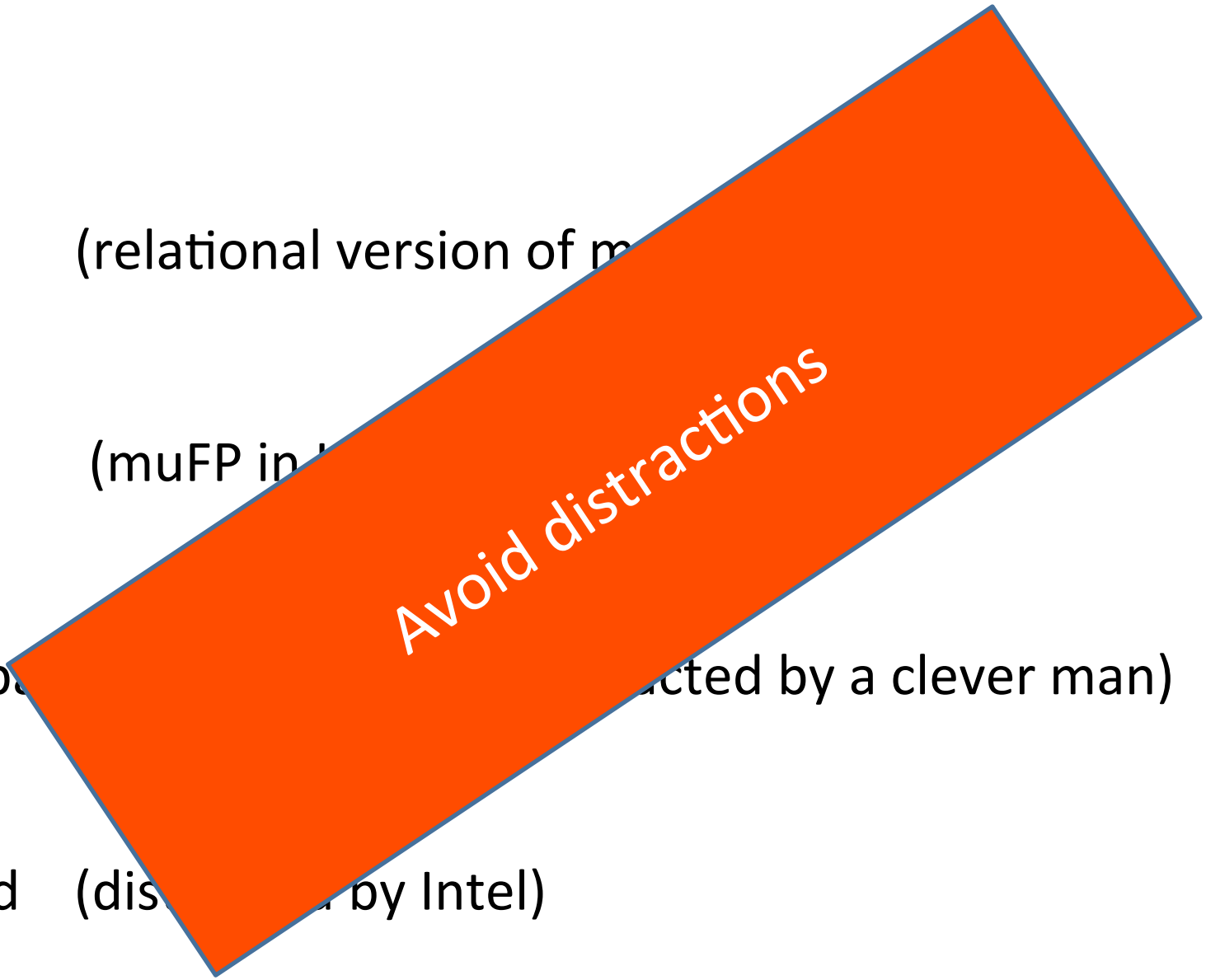
Wired (distracted by Intel)

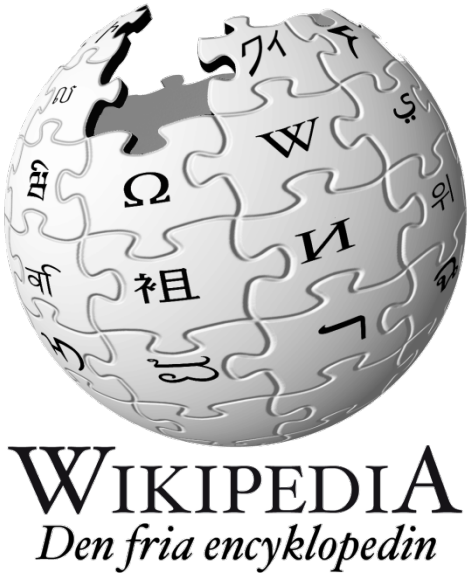
Ruby (relational version of m

Lava (muFP in

SAT-b cted by a clever man)

Wired (dis by Intel)





REALITY

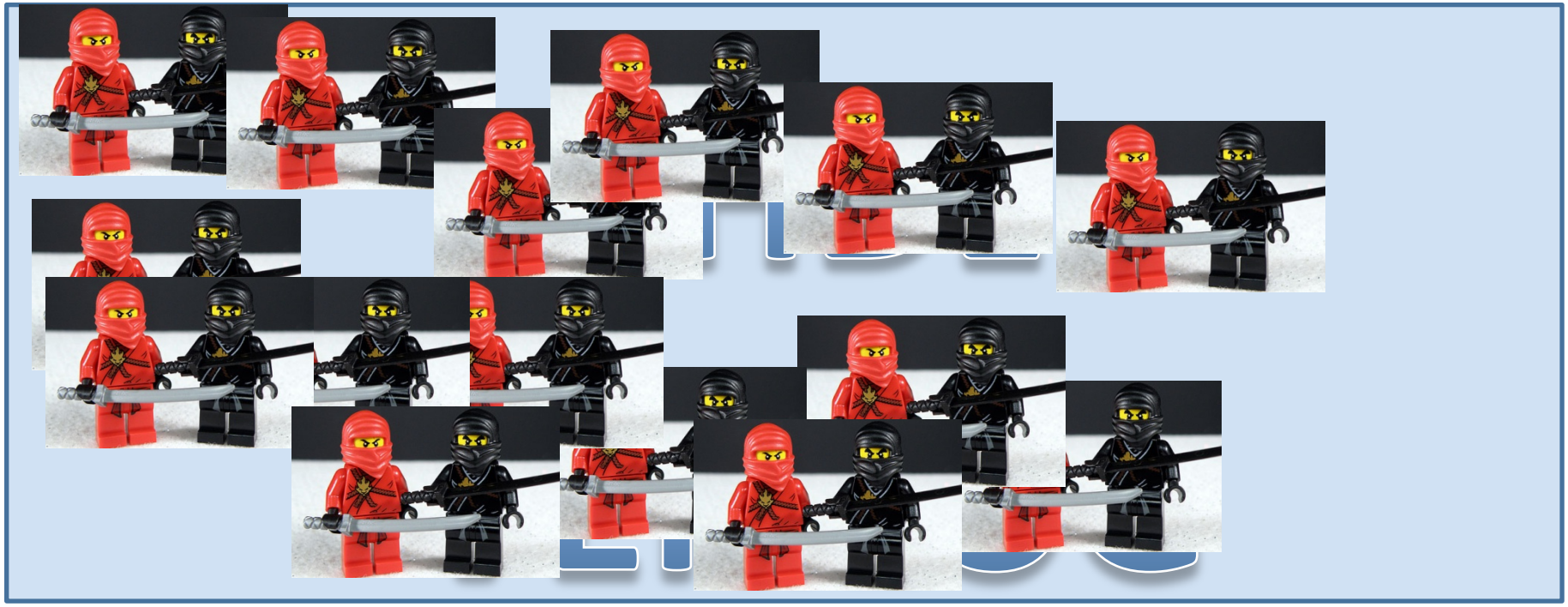
Kategori:Hårdvarubeskrivande språk
Artiklar i kategorin "Hårdvarubeskrivande språk"
Följande 2 sidor (av totalt 2) finns i denna kategori.

V

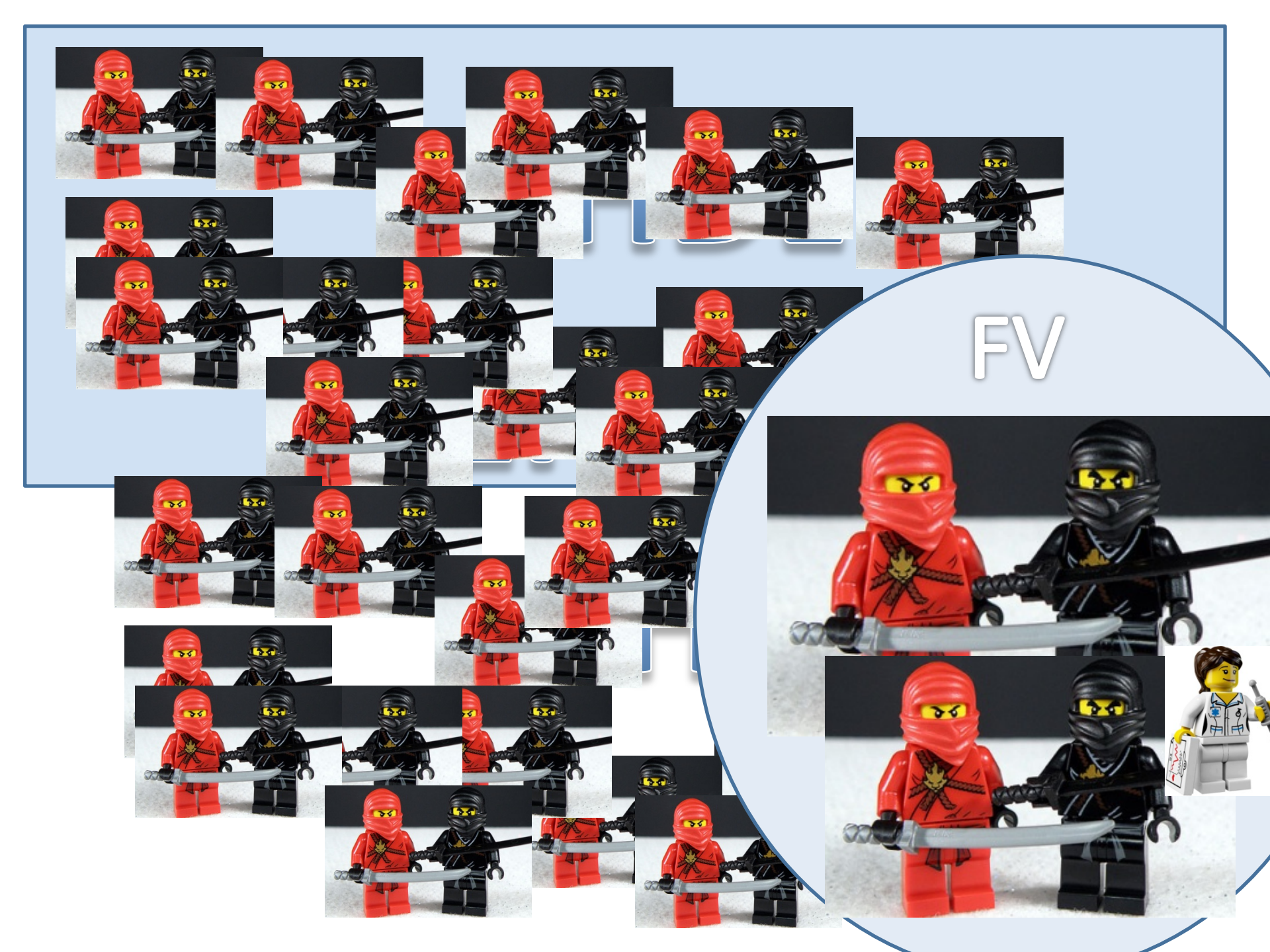
Verilog
VHDL

VHDL

VERILOG




HEROIC


FV

We failed!

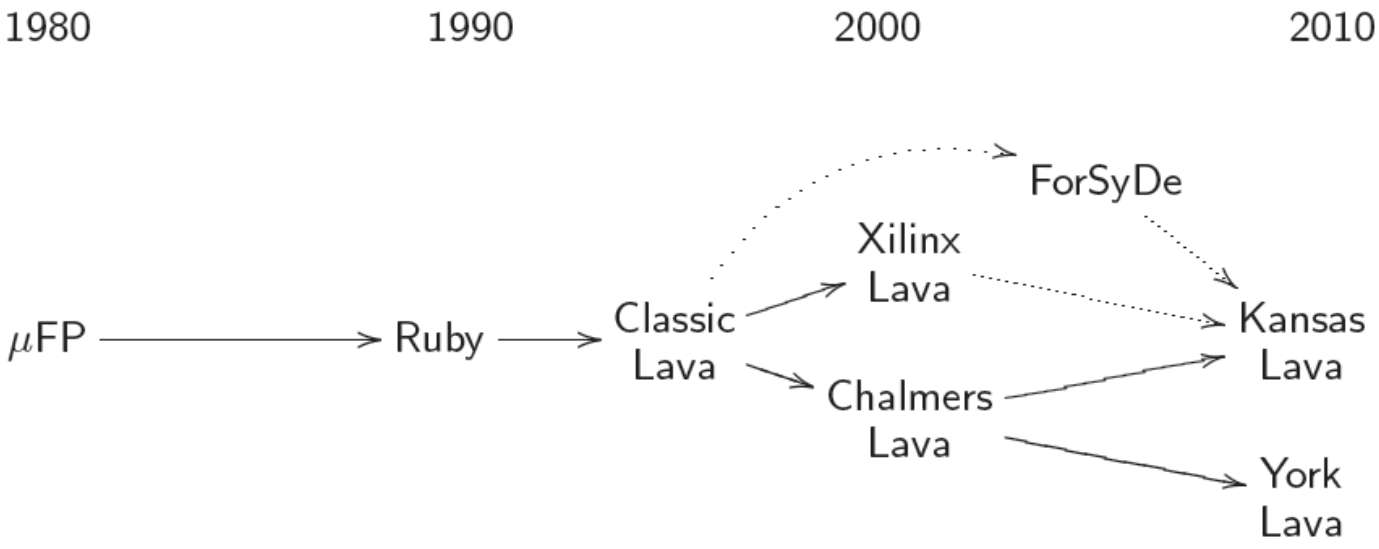
the programming languages community

We failed!

Fragmentation

Fragmentation

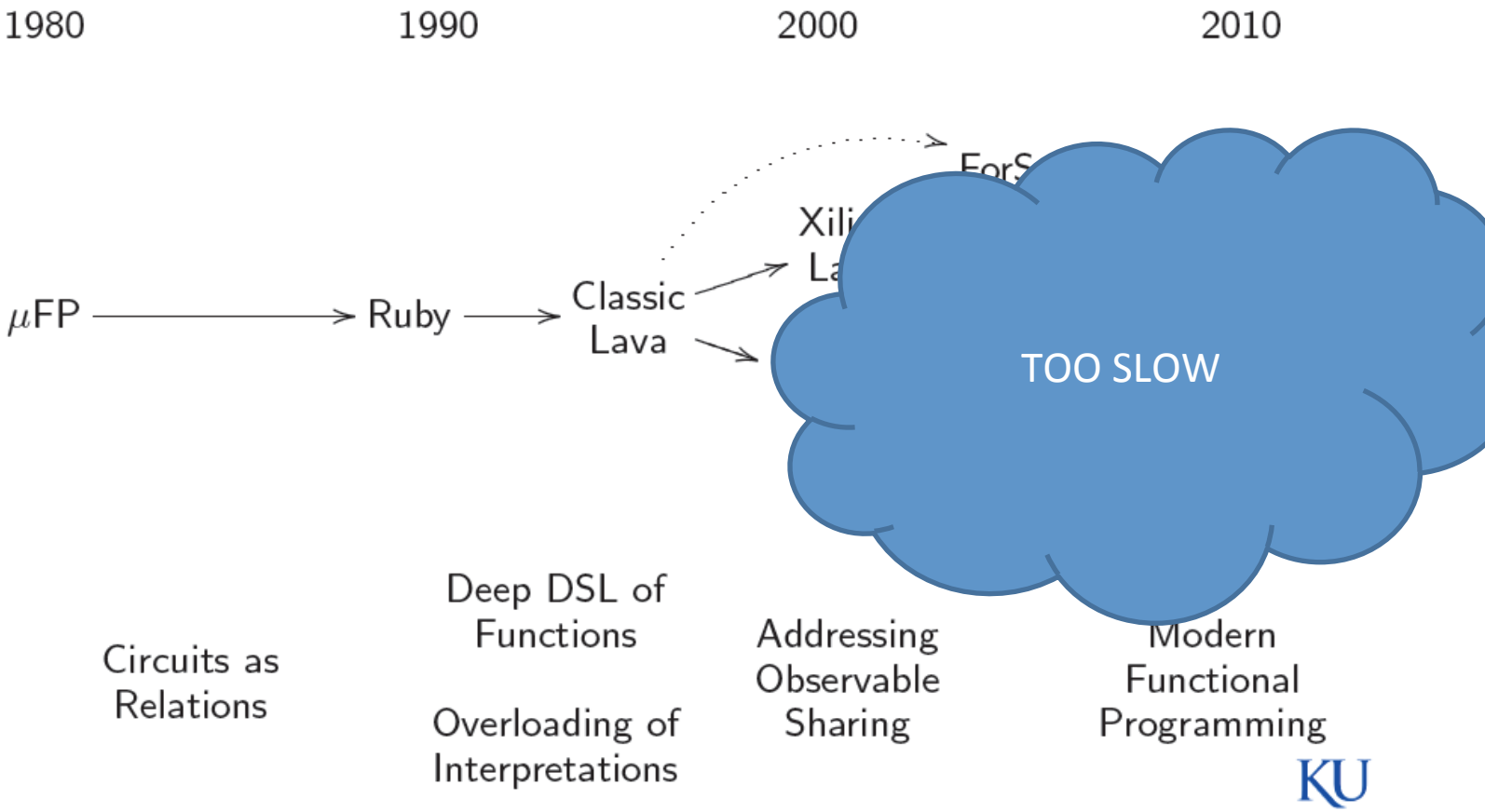
Lava History



KU

Fragmentation

Lava History



industrial contacts

necessary

fragile

have specialised problems and can distract from the real problem

SO

how can the programming languages community make amends?

SO

how can the programming languages community make amends?

parallelism

SO

how can the programming languages community make amends?

parallelism

for speed

SO

how can the programming languages community make amends?

parallelism

(not concurrency)

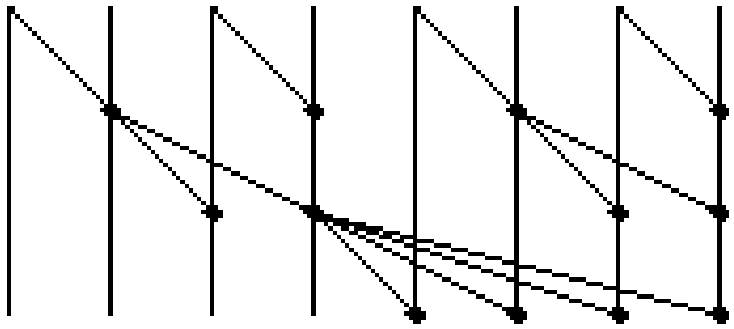
shared mutable state
is evil!

Harper: The first thing to understand is *parallelism has nothing to do with concurrency*. Concurrency is concerned with *nondeterministic composition* of programs (or their components). Parallelism is concerned with *asymptotic efficiency* of programs with *deterministic* behavior. … Parallelism, on the other hand, is all about *dependencies* among the subcomputations of a deterministic computation. The result is not in doubt, but there are many means of achieving it, some more efficient than others. We wish to exploit those opportunities to our advantage.

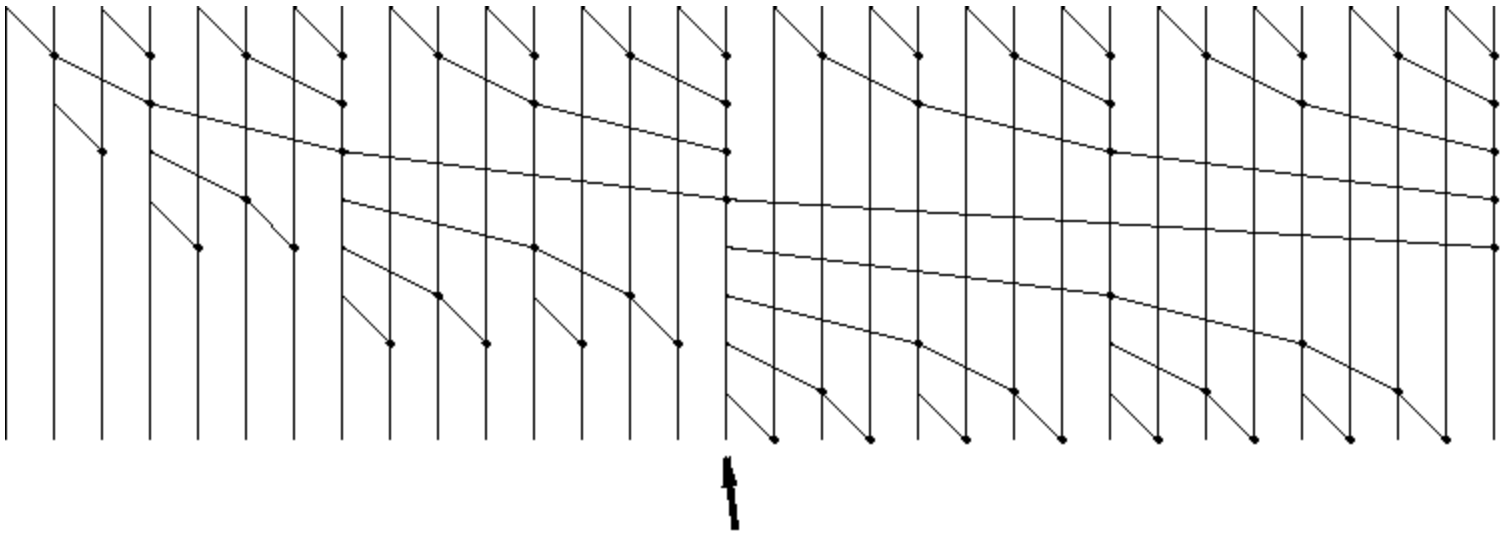
(Listen to Bob on teaching parallelism to freshmen at CMU tomorrow morning 9.00)

Marlowe, Blelloch, …

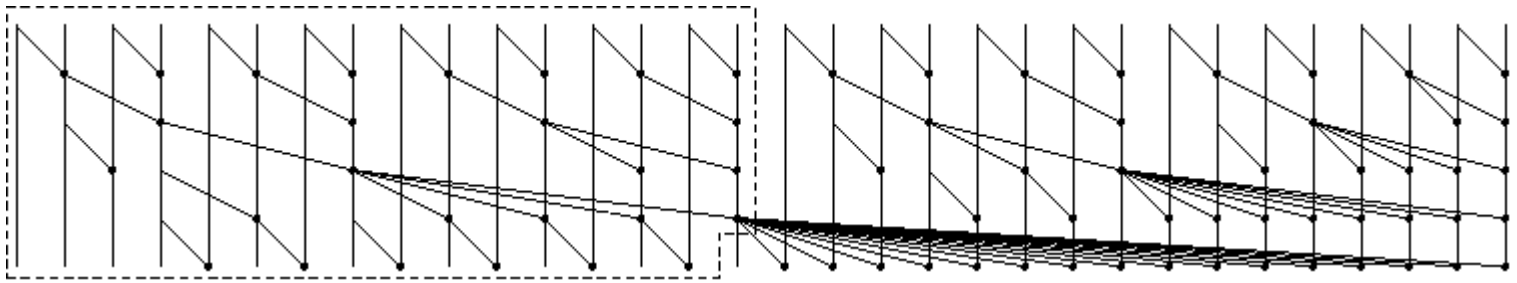
Parallel prefix (Sklansky)

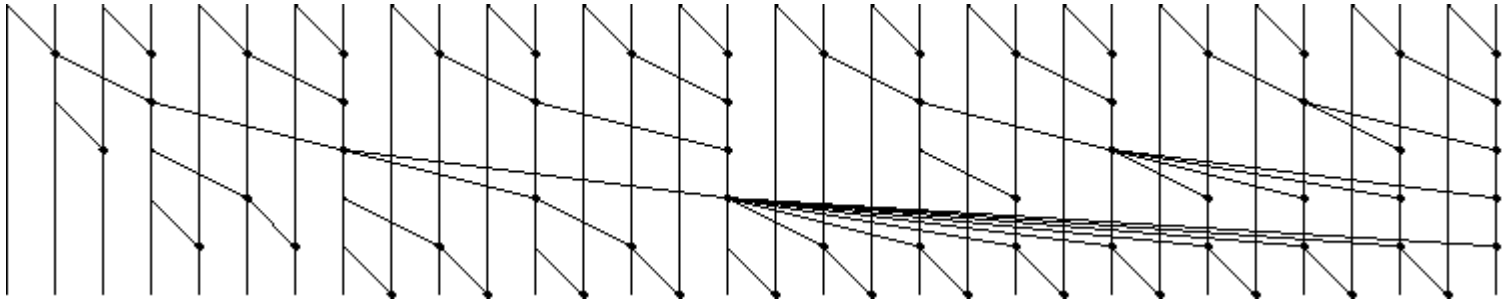


Brent Kung

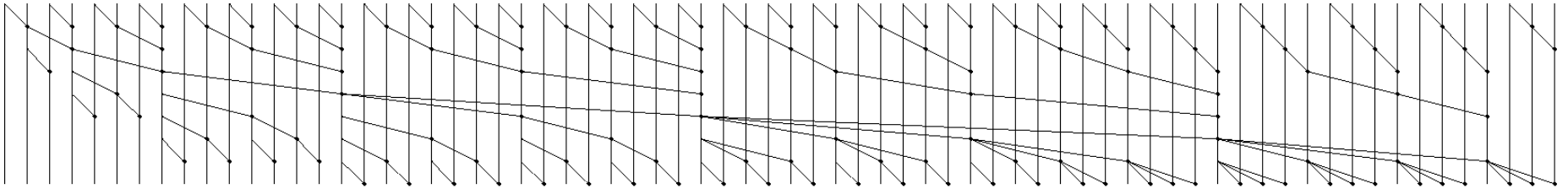


Ladner Fischer





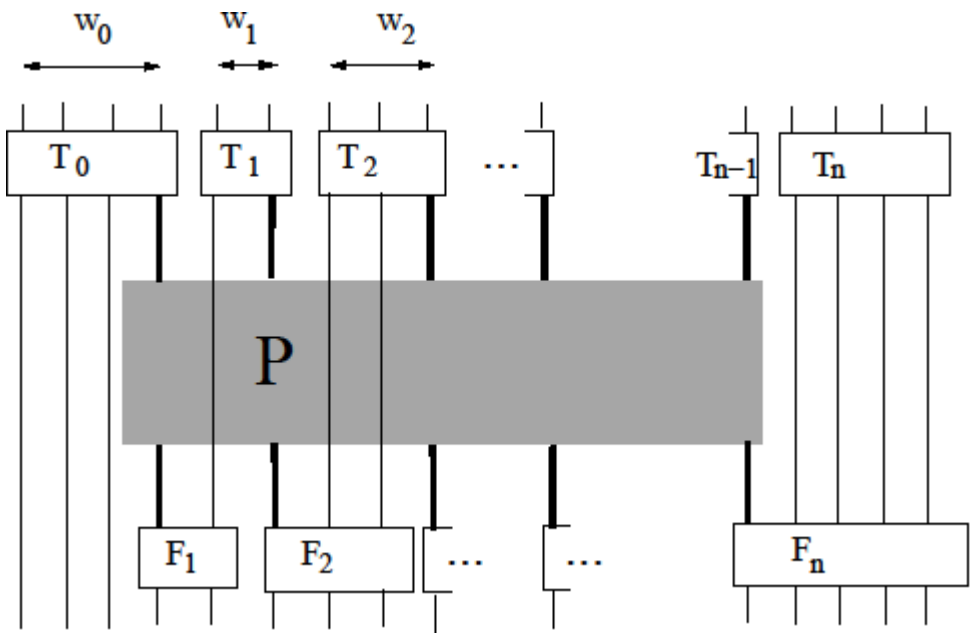
Depth size optimal (Snir)



depth + size = 2n -2

8 + 130 = 2*70 - 2

Dynamic programming + combinators



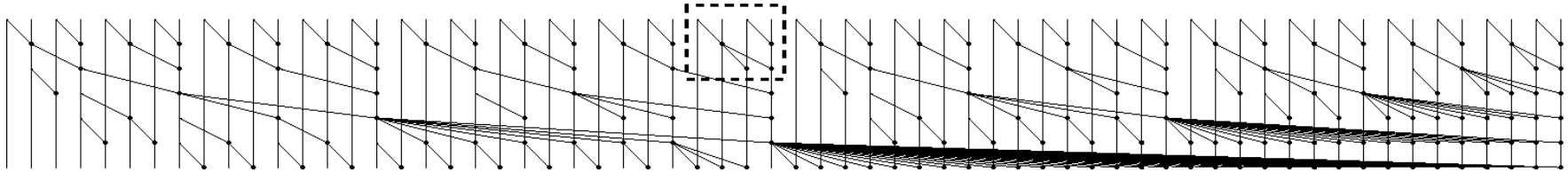
A small Haskell program can wipe out a research field ☺

Open question: smallest for minimum depth

But we can still search

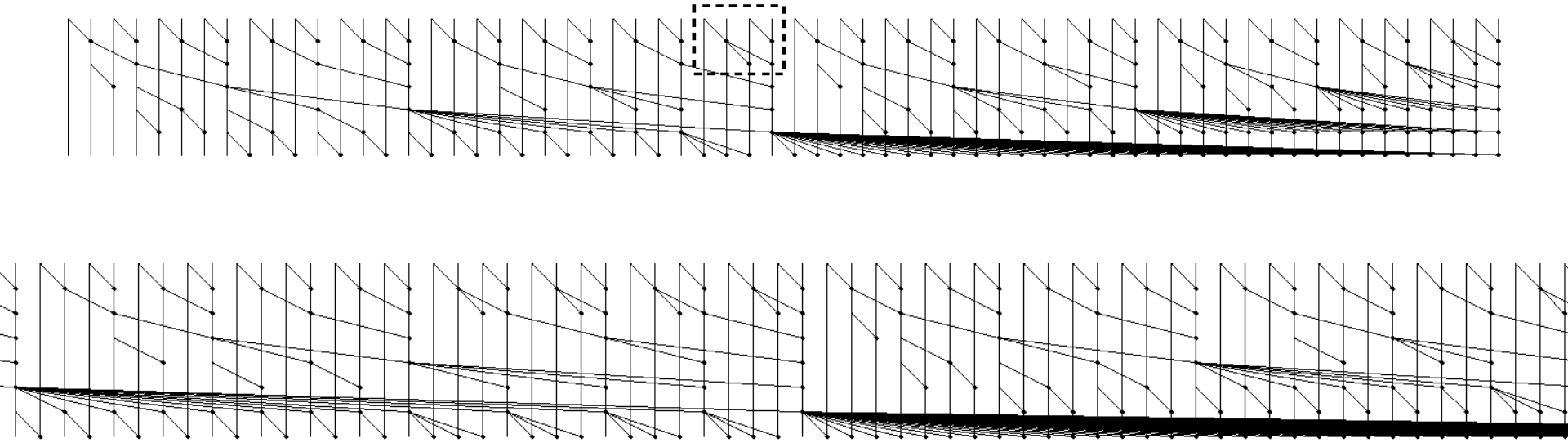
Open question: smallest for minimum depth

But we can still search



Open question: smallest for minimum depth

But we can still search

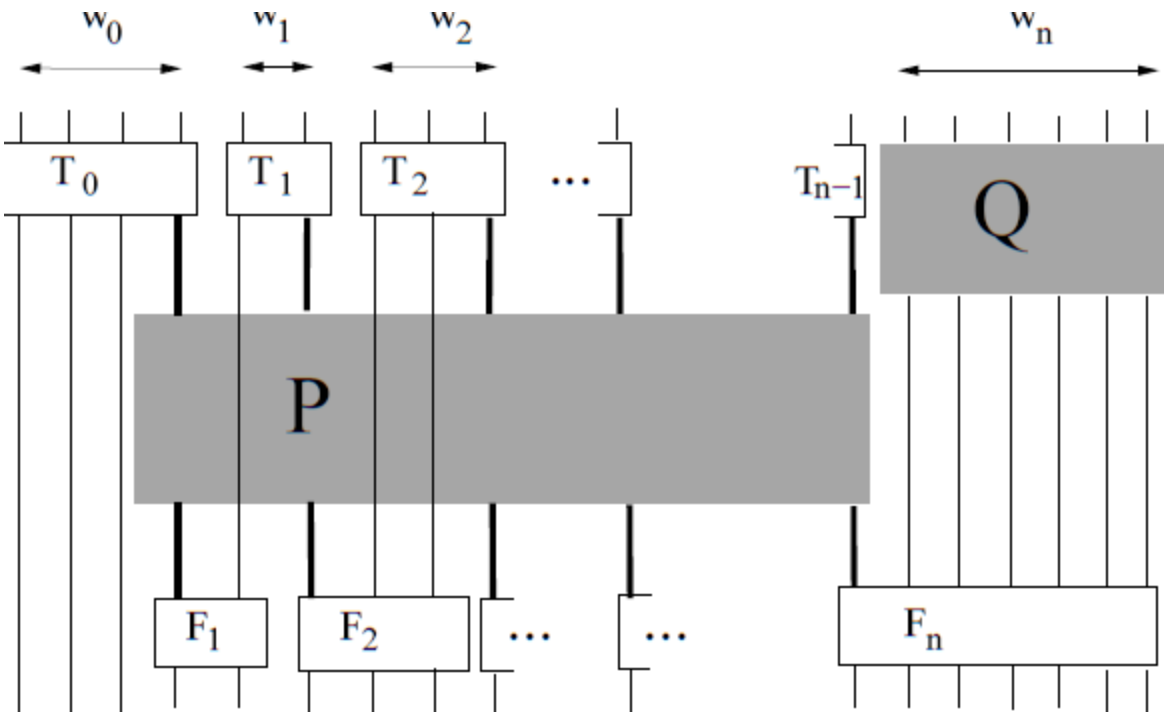


After some design exploration, the search is no longer necessary, Eureka!

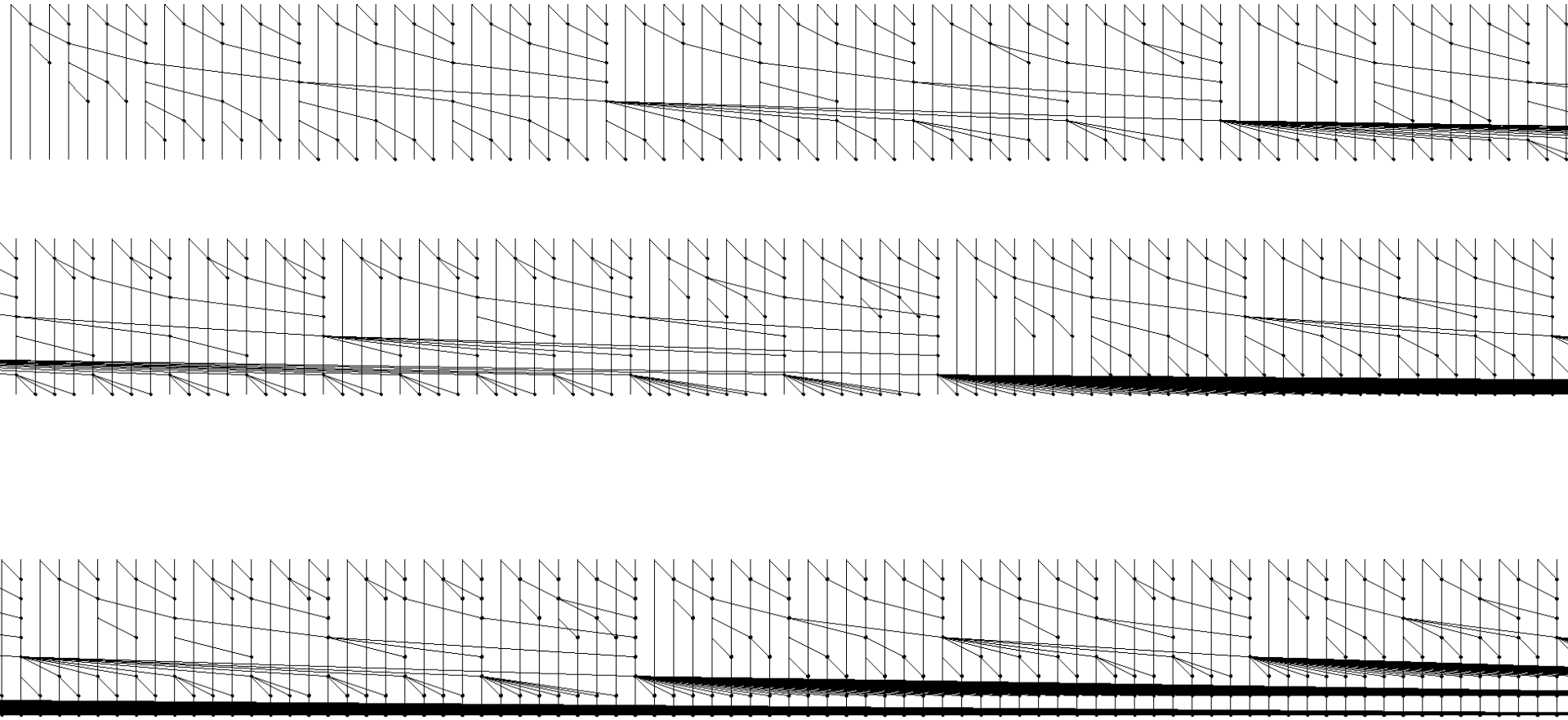
```
ppf :: Int -> PP a
ppf k = pp [1..2^k]

pp :: [Int] -> PP a
pp [_]   = wire
pp [_,_] = ser
pp is    = build2 ss (pp js) (pp (last sis))
  where
    ss  = partf is
    sis = split ss is
    js  = map last $ init sis
```

```
partf :: [Int] -> Partition
partf is = fill lis (pat (alog2 lis) ++ [r])
  where
    (lis,his) = (length is,head is)
    mid = (last is + his - 1) `div` 2
    r   = length [ k |k <- is, k > mid]

pat :: Int -> Partition
pat k | k < 6 = []
pat k = concat [replicate (2^(k-2*j-1)) (2^j) | j <- [2..(k-1) `div` 2]]

fill :: Int -> Partition -> Partition
fill k as = replicate y 2 ++ as
  where y = (k - sum as) `div` 2
```

build2



3 slices of 256 input network



Dear Mary,

Few days ago I've found your paper on constructing prefix circuits via dynamic programming.
It is interesting to note that this method allowed you to find in fact optimal complexity prefix circuits of width 2^n and depth n (at least for n<26).

I proved an exact complexity estimate 3.5*2^n-(8.5+3.5(n mod 2))2^[n/2]+n+5.
Very probably that recurrences at the end of your paper lead to the same result.

…

Igor Sergeev, Moscow State University

Dear Mary,

Few days ago I've found your paper on constructing prefix circuits via dynamic programming.
It is interesting to note that this method allowed you to find in fact optimal complexity prefix circuits of width 2^n and depth n (at least for n<26).

I proved an exact complexity estimate 3.5*2^n-(8.5+3.5(n mod 2))2^[n/2]+n+5.
Very probably that recurrences at the end of your paper lead to the same result.

…
Unfortunately I haven't English language texts to attach to the letter, all texts are in Russian, and only one short 2010 conference paper is already published.

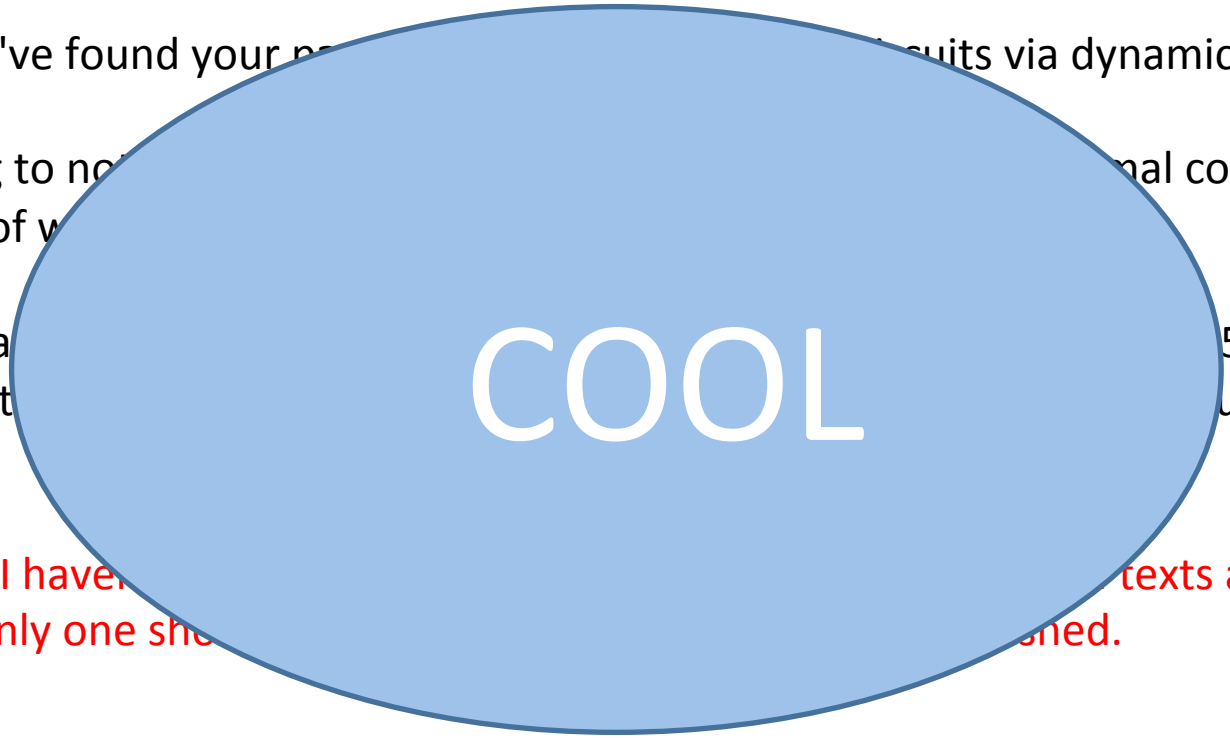
Igor Sergeev, Moscow State University

Dear Mary,

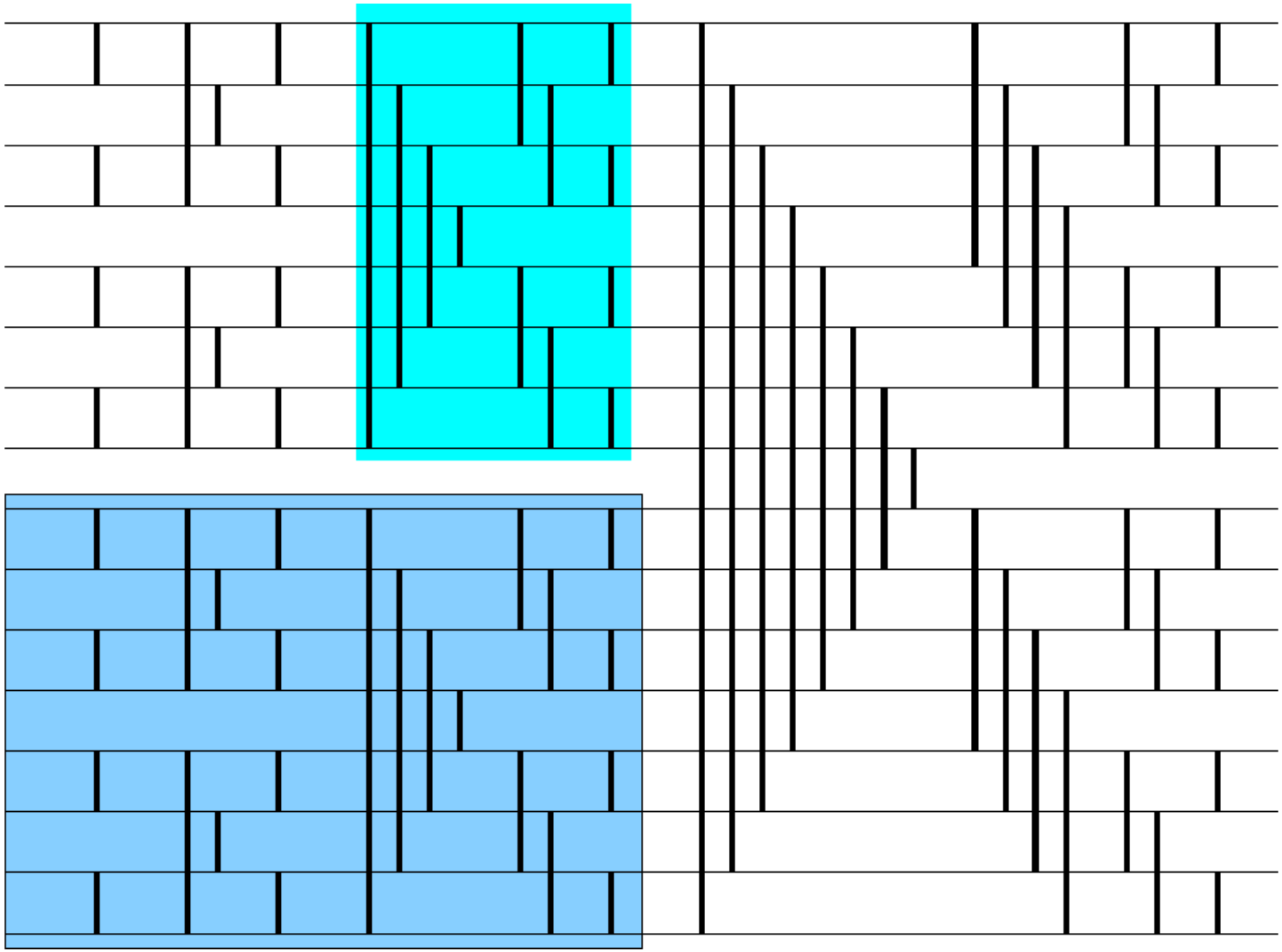
Few days ago I've found your p uits via dynamic programming.
It is interesting to no nal complexity prefix circuits of w

I proved an exa 5.
Very probably t ult.

…
Unfortunately I have texts are in Russian, and only one sh shed.

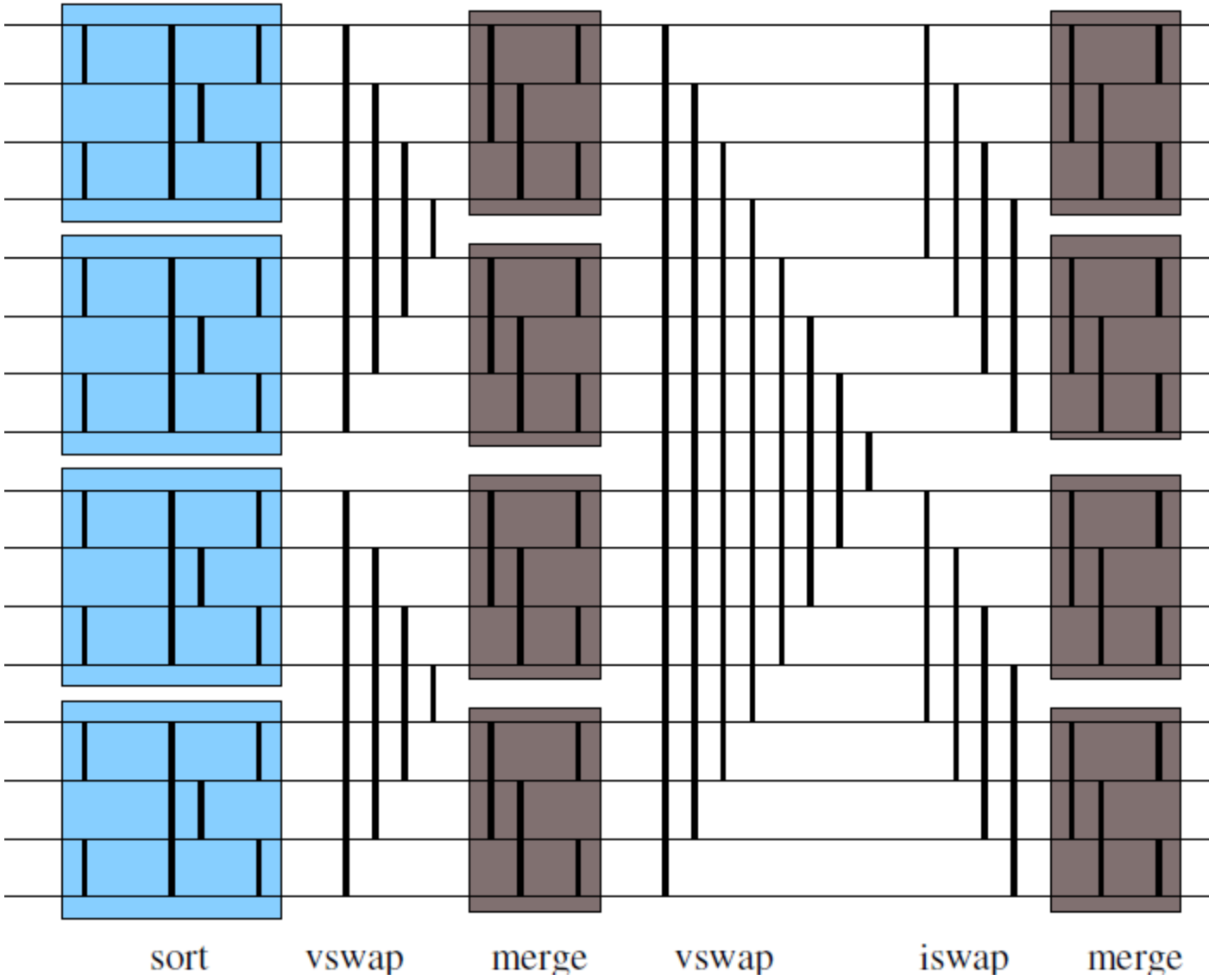


Igor Sergeev, Moscow State University



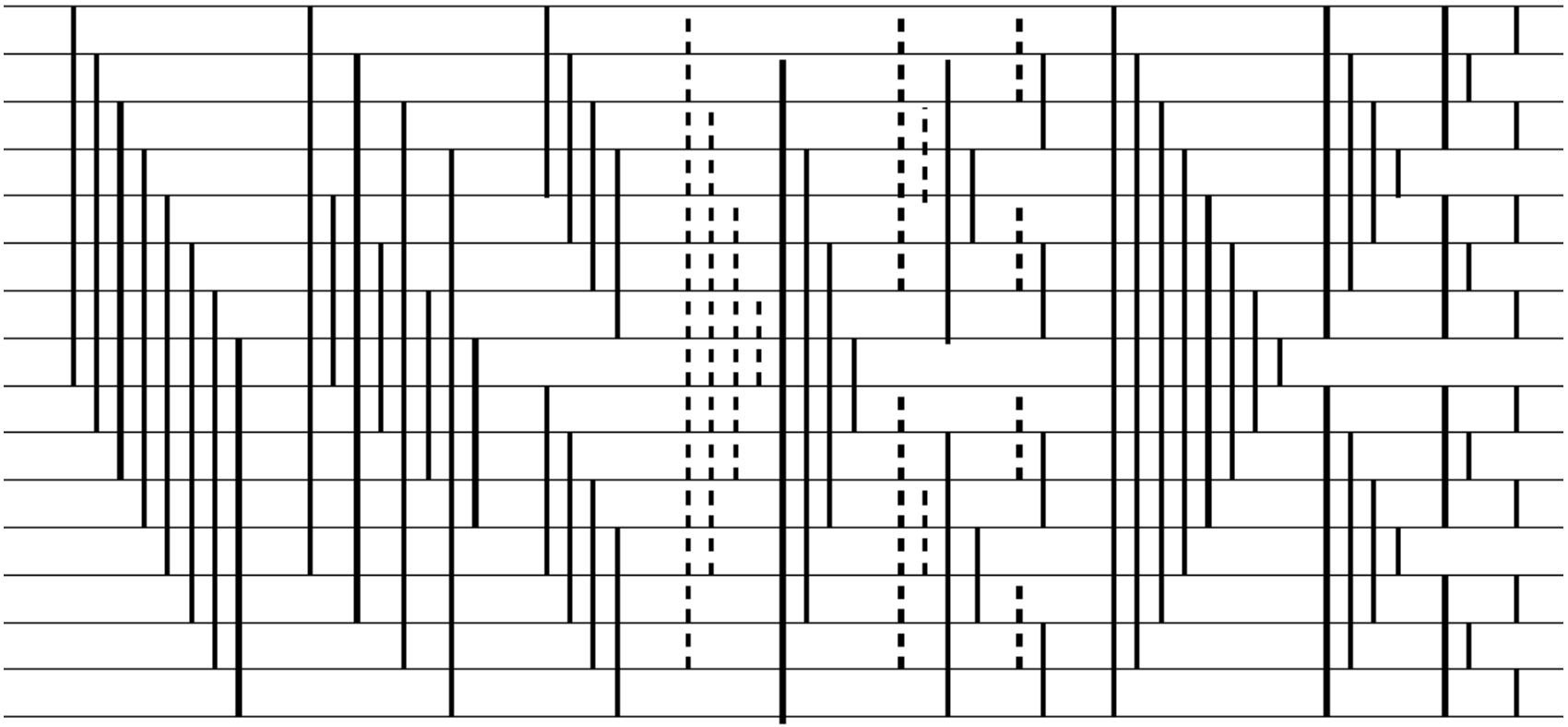
Obsidian code

```
tmerge2 :: Int -> [Array IntE -> ArrayP IntE]
tmerge2 n = vee2 (n-1) min max : [(ilv2 (n-i) min max)| i <- [2..n]]


tsort2 :: Int -> [Array IntE -> ArrayP IntE]
tsort2 n = concat [tmerge2 i | i <- [1..n]]

writes2 k = writeFile "tsort2.cu" $ CUDA.genKernel "tsort2"
                    (composeP (tsort2 k)) (namedArray "inp" (2^k))
```


sort
vswap
merge
vswap
iswap
merge

CUDA code

```
unsigned int arrayLength = 1 << LOG_L_SIZE;
unsigned int diff = LOG_L_SIZE - LOG_S_SIZE;
unsigned int blocks = arrayLength / S_SIZE;
unsigned int threads = S_SIZE / 2;

sortSmall<<<blocks, threads,4096>>>(din,din);

for(int i = 0 ; i < diff ; i += 1){
  vSwap<<<blocks/2,threads*2,0>>>(din,din,(1<<i)*S_SIZE);

  for(int j = i-1; j >= 0; j -= 1)
    iSwap<<<blocks/2,threads*2,0>>>(din,din,(1<<j)*S_SIZE);

  bmergeSmall<<<blocks,threads,4096>>>(din,din);}
```



CUDA code

```
unsigned int arrayLength = 1 << LOG_L_SIZE;
unsigned int diff = LOG_L_SIZE - LOG_S_SIZE;
unsigned int blocks = arrayLength / S_SIZE;
unsigned int threads = S_SIZE / 2;

sortSmall<<<blocks, threads,4096>>>(din,din);

for(int i = 0 ; i < diff ; i += 1){
  vSwap<<<blocks/2,threads*2,0>>>(din,din,(1<<i)*S_SIZE);

  for(int j = i-1; j >= 0; j -= 1)
    iSwap<<<blocks/2,threads*2,0>>>(din,din,(1<<j)*S_SIZE);

  bmergeSmall<<<blocks,threads,4096>>>(din,din);}
```

CUDA code

```
for(int j = i-1; j >= 0; j -= 3){
  if (j==0)
    iSwap<<<blocks/2,threads*2,0>>>(din,din,(1<<j)*S_SIZE);
    else
    {if (j==1)
       iSwap2<<<blocks/4,threads*2,0>>>(din,din,(1<<j)*S_SIZE);
       else
       iSwap3<<<blocks/8,threads*2,0>>>(din,din,(1<<j)*S_SIZE);}}
```

result is fast enough

Sorter (without warp size related optimisations) is pleasingly fast (three times faster than the NVIDIA bitonic code in SDK
but it sorts key-value pairs)

factor of 30 over a single (weedy) CPU

result is fast enough

Sorter (without warp size related optimisations) is pleasingly fast (three times faster than the NVIDIA bitonic code in SDK
but it sorts key-value pairs)

factor of 30 over a single (weedy) CPU

Can get a lot faster too… Aim to think about the algorithm and then decompositions. NOT THREADS.

(just GPUs + declarative)

Obsidian Claessen, Sheeran, Svensson

Nikola Mainland, Morrisett

Accelerate Chakravarty et al

Hiperfit work ??

DDAs Burrows, Haveraaen

Ypnos Orchard, Bollingbroke, Mycroft

Accelerator Singh

many more

(just GPUs + Haskell)

Obsidian Claessen, Sheeran

Nikola Mainland

Accelerate Chakravarty, Keller, Lee, McDonell, Grover

Hiperfit work ??

DDAs Burrows, Henriksen, Oancea

Ypnos, Orchard, Bolingbroke, Mycroft

Accelerator Singh

many more

We should collaborate more!

DSL problems

From [Kosar et al, 2008]:

“The main disadvantage of DSLs is the cost of their development.”

DSL problems

From [Kosar et al, 2008]:

"The main disadvantage of DSLs is the cost of their development."

"This is one of the reasons why DSLs are rarely used in solving software engineering problems."

DSL problems

From [Kosar et al, 2008]:

"The main disadvantage of DSLs is the cost of their development."

"This is one of the reasons why DSLs are rarely used in solving software engineering problems."

Also: lack of efficiency of generated code (esp. for embedded real-time software)

Example DSL construct: *data-parallel arrays*

$$a_i = f(i), \quad 0 \le i < len$$

- *Independent* computation of array elements based on index
- Common operation in numeric array processing
- Amenable to optimization (loop fusion)
- Parallelizable

EDSL constructs in Haskell

- No need to extend host-language syntax
- Custom-looking syntax obtained by *higher-order functions*

```
parallel len (\i -> f i)
```

- **`parallel`** is an ordinary Haskell function that takes another function as argument

Syntactic – a Haskell library for EDSL implementation

- Even embedded languages are costly to implement
- Similar-looking constructs reimplemented over and over again
  - Across different languages
  - But also within a single language

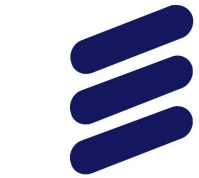
Syntactic features

- A generic abstract syntax tree customizable for different domains
- Generic implementations of common syntactic constructs (e.g. conditionals, tuples, variable binding)
- Generic semantic interpretations and transformations
- Available as a Haskell package: http://hackage.haskell.org/package/syntactic

Feldspar

- Functional look-and-feel, targetting C for DSP processors
  - manycore is the trigger
- Much more concise, good performance (in some small examples, early days)
- Developed in cooperation by Ericsson, Chalmers University and ELTE University (open source)
- Syntactic grew out of the Feldspar implementation
- More info: http://feldspar.inf.elte.hu/feldspar/

- ”Feldspar lets me think big thoughts”
- ”Feldspar lets me have my cookie and eat it too”



ERICSSON

Feldspar example

```
prog a =
    parallel 10 (\i ->
        i+(a*2)
    )
```

Feldspar example

```
prog a =
    parallel 10 (\i ->
        i+(a*2)
    )
```

```
*Main> eval prog 5
[10,11,12,13,14,15,16,17,18,19]
```

Evaluation in the Haskell interpreter

Feldspar example: generated C code

```
prog a =
    parallel 10 (\i ->
        i+(a*2)
    )
```

```
*Main> icompile prog
```

```
void test(struct array mem, uint32_t v0, struct array * out)
{
    uint32_t v2;

    v2 = (v0 << 1);
    for(uint32_t v1 = 0; v1 < 10; v1 += 1)
    {
        at(uint32_t,(* out),v1) = (v1 + v2);
    }
}
```

Invariant code hoisting

Feldspar example: syntax tree

```
prog a =
    parallel 10 (\i ->
        i+(a*2)
    )
```

```
*Main> drawAST prog
```

```
Lambda 0
|
`- Let 2
   |
   +- (*)
   |  |
   |  +- var:0
   |  |
   |  `- 2
   |
   `- parallel
      |
      +- 10
      |
      `- Lambda 1
         |
         `- (+)
            |
            +- var:1
            |
            `- var:2
```

Feldspar implementation

Assembling the language:

```
type FeldDomain
    =   Condition    TypeCtx
    :+: Let TypeCtx TypeCtx
    :+: Literal      TypeCtx
    :+: Select       TypeCtx
    :+: Tuple        TypeCtx
    :+: Array
    :+: BITS
    :+: COMPLEX
    :+: NUM
    ...

type FeldDomainAll = HODomain TypeCtx FeldDomain

newtype Data a = Data (AST FeldDomainAll (Full a))
```

Constructs reused from Syntactic

Feldspar-specific constructs

Main Feldspar expression type

Generic AST type specialized for Feldspar

Data-parallel arrays, implementation

Syntax, type and semantics in 9 lines:

```
data Array a where
    Parallel :: Type a => Array
      (Length :-> (Index -> a) :-> Full [a])
    ...

instance IsSymbol Array where
    toSym Parallel = Sym "parallel"
        (\len ixf -> genericTake len $ map ixf [0..])
    ...

parallel :: Type a =>
    Data Length -> (Data Index -> Data a) -> Data [a]
parallel = sugarSym Parallel
```

Expressing semantics using ordinary Haskell functions

Exposed to the user

(Leaving out ≈ 10 lines of mechanical declarations)

Data-parallel arrays, implementation

What do we get?

- Strongly typed abstract and concrete syntax
- Interpretations:
  - Evaluation
  - Syntax tree rendering, etc.
- Transformations
  - Constant folding
  - Common sub-expression elimination
  - Invariant code hoisting, etc.
- Caveat: Assuming functional semantics of the DSL

Data-parallel arrays, implementation

Code generation:

```
compileProgSym Parallel _ loc
  (len :*: (Symbol (Ann info lam) :$: ixf) :*: Nil)
  | Just (Lambda v) <- prjCtx typeCtx lam
  = do let ix@(Var _ name) =
             mkVar (compileTypeRep $ argType $ infoType info) v
       len' <- compileExpr len
       (e, body) <- stealProg $ compileProg (loc :!: ix) ixf
       tellProg [For name len' 1 $ Seq body]
```

Compiled to an imperative for loop (for now)

Recursive compilation of body

Extension and experimentation

```
type FeldDomain
    =   Condition   TypeCtx
    :+: Let TypeCtx TypeCtx
    :+: Literal     TypeCtx
    :+: Select      TypeCtx
    :+: Tuple       TypeCtx
    :+: Array
    :+: BITS
    :+: COMPLEX
    :+: NUM
    ...

type FeldDomainAll = HODomain TypeCtx FeldDomain

newtype Data a = Data (AST FeldDomainAll (Full a))
```

- Constructs developed independently and assembled *"in the last minute"*
- Simplifying cooperation and experimentation
- Allowing proprietary extensions?

Summary

- Embedding is an efficient DSL implementation technique
- Syntactic greatly simplifies EDSL implementation (at least for functional DSLs)
  - Focus on the truly domain-specific parts
  - Reuse the rest!

- Embedding
  technique
- Syntactic g
  (at least fo
  - Focus on the truly domain-specific parts
  - Reuse the rest!

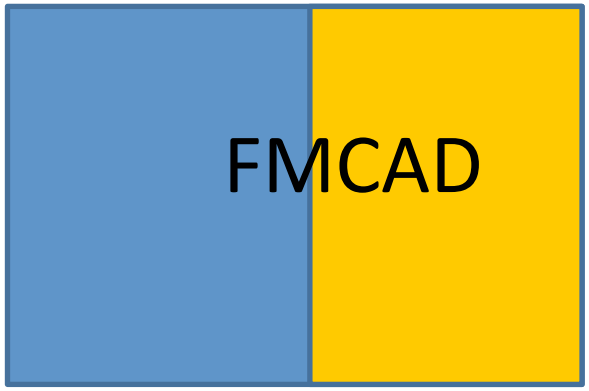
Complete implementation of parallel arrays in Feldspar:
< 50 lines of code!

Related work

- Language workbenches [Fowler, 2005] [Kats et al 2010]
  - Declarative specification of editing environments and code generators
- Embedding in Scala [Chafi et al, 2010]
  - Direct overloading of host language constructs

(see slide at end)

Learn



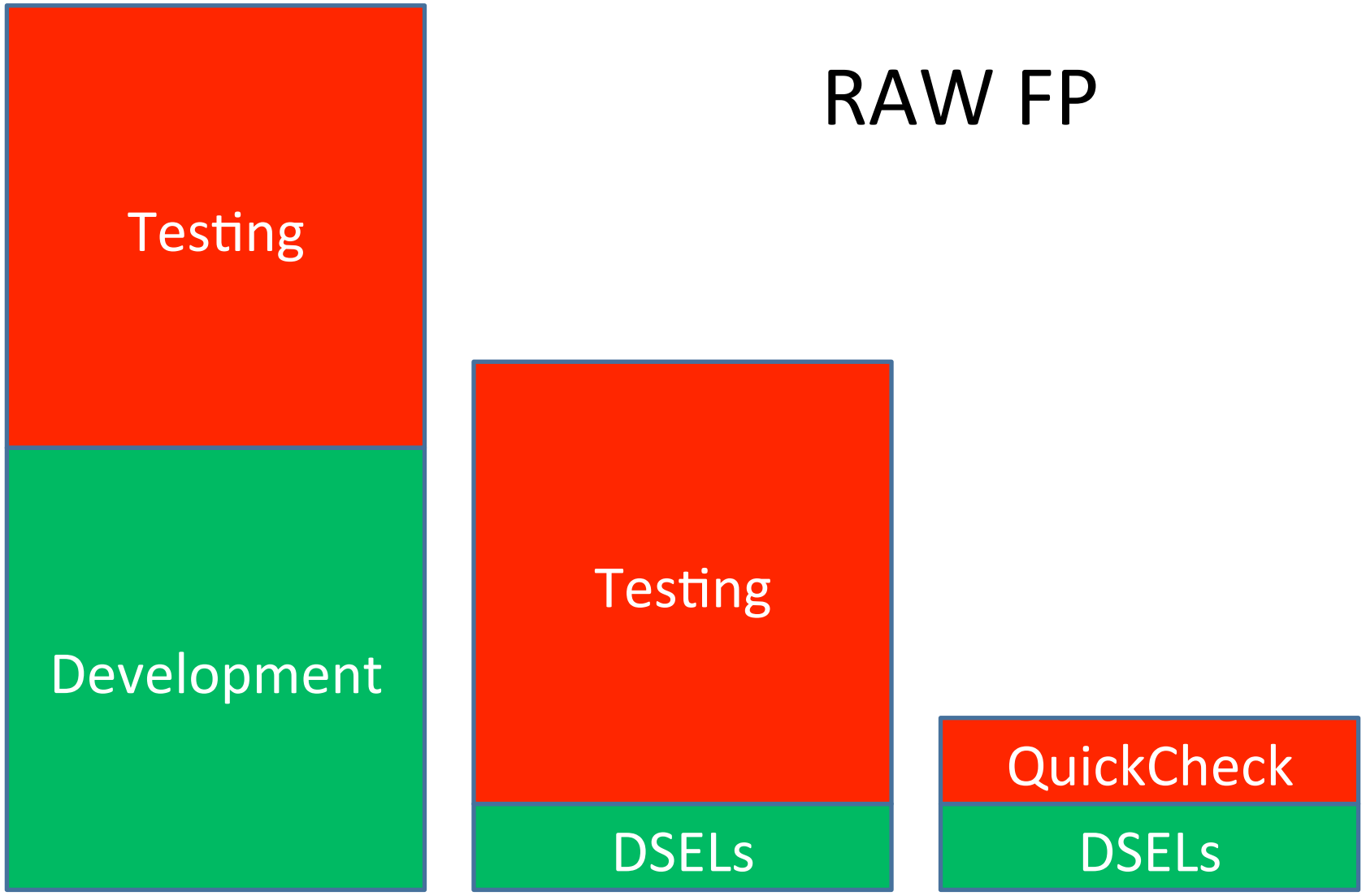
Learn

FMCAD

ICFP

CUFP

RAW FP



HIPERFIT

- Det Strategiske Forskningsråd har investeret 31,4 mio. kr. i det forskningscenter, der skal udvikle computerne, som har fået navnet HIPERFIT - functional high-performance computing for financial information technology. Bag projektet står bl.a. Københavns Universitet, Danske Bank, Jyske Bank, Nordea, Nykredit, SimCorp og den franske it-virksomhed Lexifi samt en række internationale forskningspartnere bl.a. University of Oxford, University of California og Carnegie Mellon.

- Gorm Praefke betegner den nye teknologi som en nødvendighed for den finansielle branche.

Maybe we need to make a conference and grow a community??

Conclusion

Programming languages (DSLs) are the answer to nearly everything!

The group of people at the Hiperfit worksop has what it takes to deliver this time

But we need AMBITIOUS CONCRETE GOALS if we are to help make parallel programming productive in reality

One paper at a time won't cut it

References

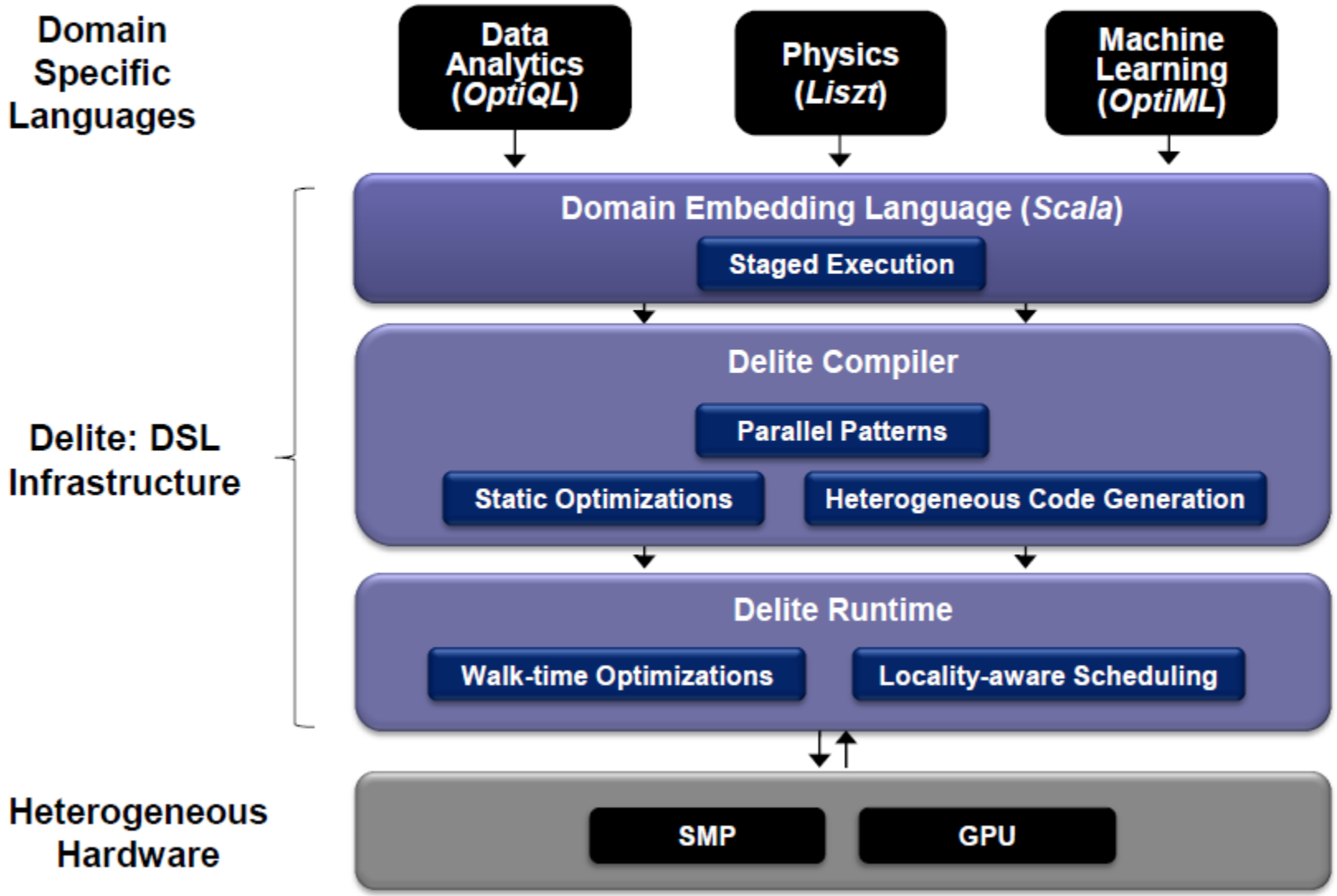
Kosar et al. *A preliminary study on various implementation approaches of domain-specific language*. Information and Software Technology. 2008. Elsevier.

Martin Fowler. *Language Workbenches: The Killer-App for Domain Specific Languages?* 2005. http://www.martinfowler.com/articles/languageWorkbench.html

Kats et al. *The Spoofax Language Workbench. Rules for Declarative Specification of Languages and IDEs*. OOPSLA, ACM, 2010.

Chafi et al. *Language virtualization for heterogeneous parallel computing*. OOPSLA, ACM, 2010.

Delite Overview



A Heterogeneous Parallel Framework for Domain-Specific Languages, Brown et al , PACT 11 (slide from associated talk)

CS and SE people mean the same by DSL!!!
(but SE people seem not have noticed
modern functional host languages)

Kosar et al. *A pr*
domain-spec
Elsevier.

Martin Fowler. *Language Workbenches: The Killer-App for Domain Specific Languages?* 2005. http://www.martinfowler.com/articles/languageWorkbench.html

Kats et al. *The Spoofax Language Workbench. Rules for Declarative Specification of Languages and IDEs*. OOPSLA, ACM, 2010.

Chafi et al. *Language virtualization for heterogeneous parallel computing*. OOPSLA, ACM, 2010.