Kestrel Institute

Specware™

John Anton

Kestrel Institute

March 29, 2001

Suppose

- … you could compute solutions to one of the world’s largest and most complex computational problems

  by filling in some blanks in a table
- … your code ran up to 100 times faster (and more) than typical schedulers for the same problems
- … you could prove the solutions are correct
- … you could change your system in a fraction of the time required for today’s practices

Computer Aided Mission Planner (CAMPS)

Well, the Air Force could.

The Air Force Mobility Command will field CAMPS with a capability like that by the end of 2001.

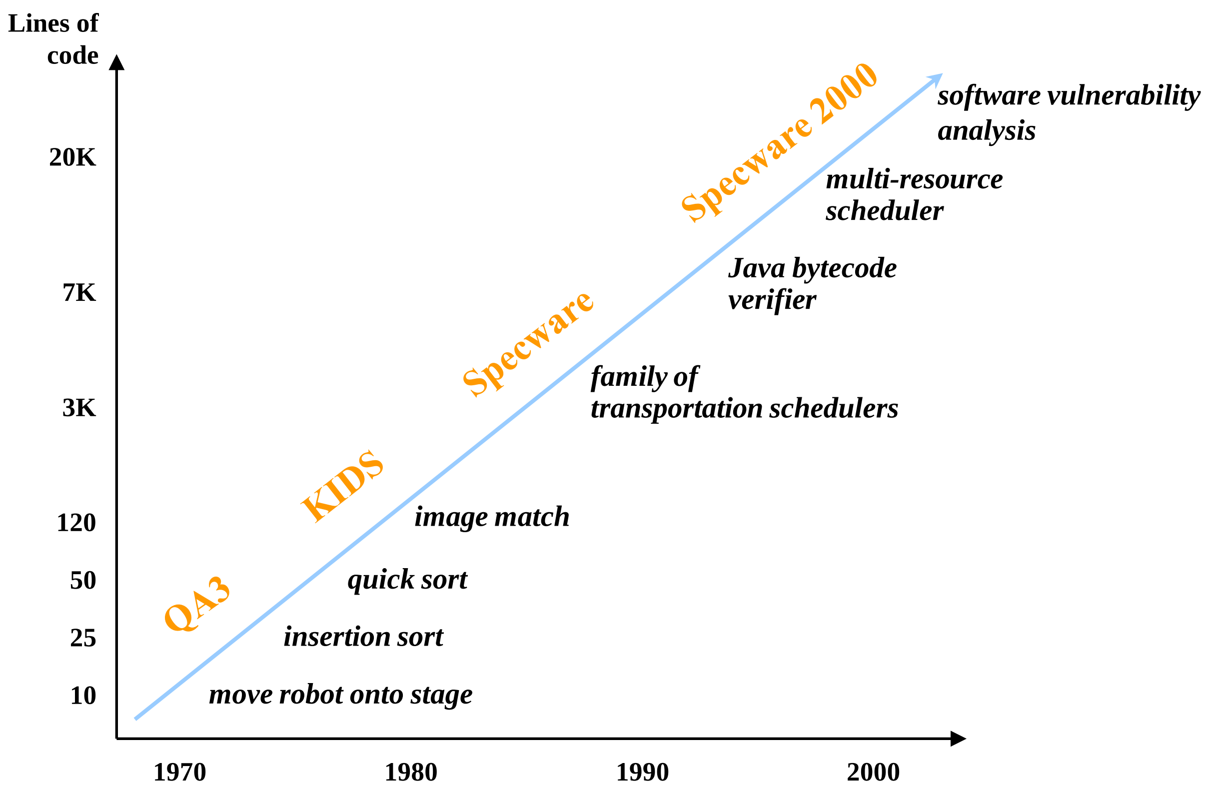
CAMPS is built with an **generator** based on Kestrel Institute's software synthesis technology.

Kestrel’s mission is wide spectrum

- Conduct basic computer science research
- Build prototype applications
- Transfer technology
- Incubate technology
- Spin-out technology companies

…and our focus on program synthesis is obsessive

30 year technology path



Presentation Outline

- What is Specware?
- What can you do with it?
- How do you use it?
- How does it supply leverage?
- Who has used it?
- How do you work with us?

Specware provides an environment…

…for **high assurance computing**

…by means of **clear expression of design theories**

…and for those theories

- …a computing framework to evaluate them
- …tools for refining them
- …inference systems to reason about them
- …semantic rigor to trust them

What can you do with Specware?

- **Build specifications**
  - Write, parse, compile, save
  - Combine, reuse, revise, import, refine
  - Parametrize on other specifications
- **Refine specifications to code**
  - Specware 2000, Boeing equipment layout, etc.
  - Lisp, C (under re-construction), Java (under development)
- **Prove correctness of the specifications**
  - Choice of several provers: Snark, Gandalf, other
- **Analyze specifications and code**
  - Motorola AIM
  - Java BCV, SVA (NSA), MoBIES (DARPA)

How do you build specifications?

- Use the basic Specware language to:
  - Introduce sorts, constants, definitions, operations
  - Express axioms to restrict the behavior of components
  - Build a domain specification language
- Get system help for:
  - Defining, importing, and revising specifications
  - Composing and refining specifications
  - Proving theorems about specifications
- Use built-in automation to:
  - Assist construction of composite specifications from their component modules

Example specification from bootstrapping Specware

spec Category =

import translateSpec ReflexiveGraph

["Node" $\mapsto$ "Obj",

"Edge" $\mapsto$ "Arr"]

sort Composable = { (f, g) : Arr $\times$ Arr | dom(f) = cod(g) }

op compose : Composable $\rightarrow$ Arr

axiom dom-compose **is** $\forall$ (f, g) (dom $\circ$ compose)(g,f) = dom(f)

axiom cod-compose **is** $\forall$ (f, g) (cod $\circ$ compose)(g,f) = cod(g)

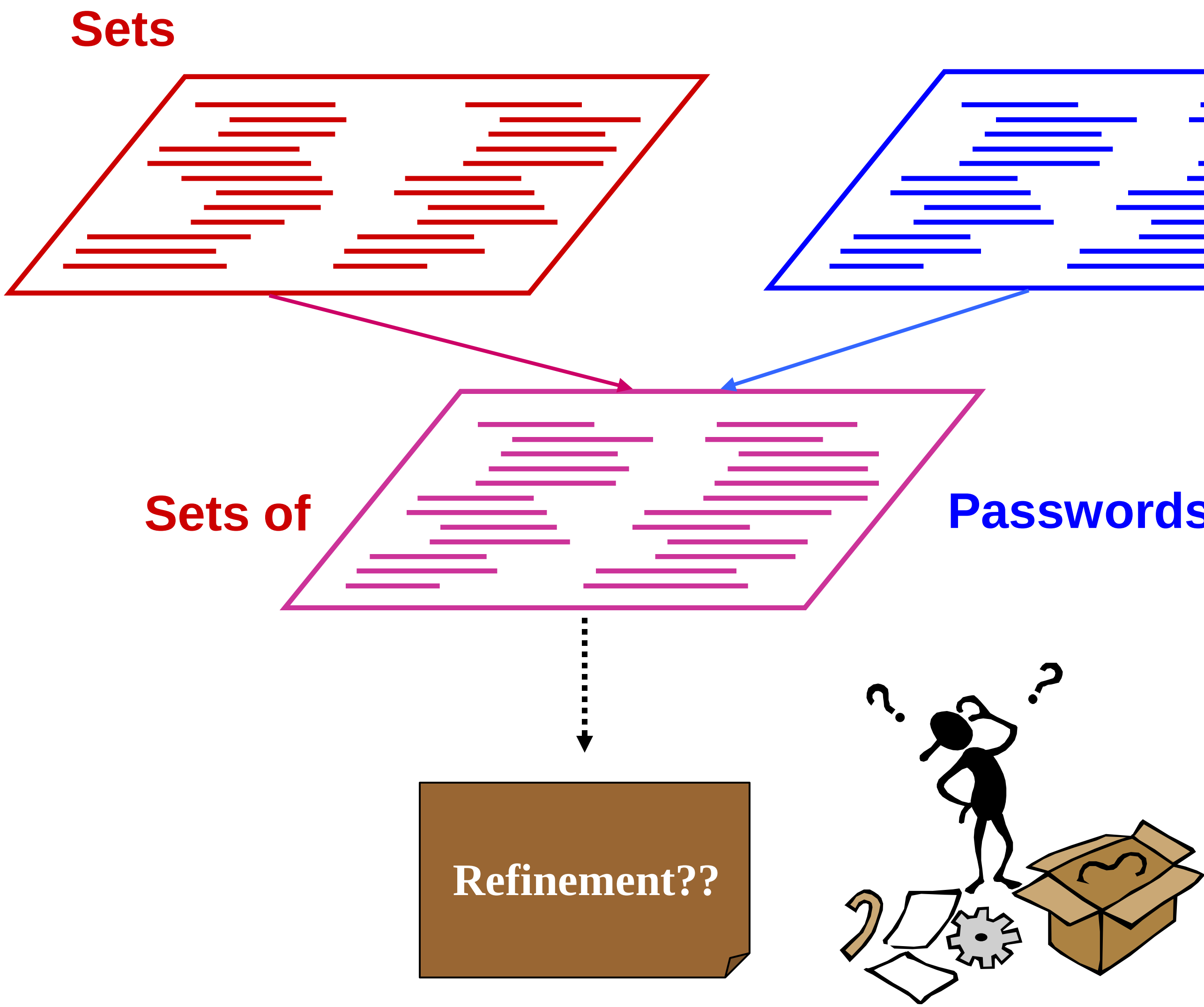
axiom assoc **is** $\forall$ (f, g, h) compose(h, compose(g,f)) = compose(compose(h, g), f)

axiom r-unit **is** $\forall$ (f) compose(f, (ident $\circ$ dom) f) = f

axiom l-unit **is** $\forall$ (f) compose((ident $\circ$ cod) f, f) = f

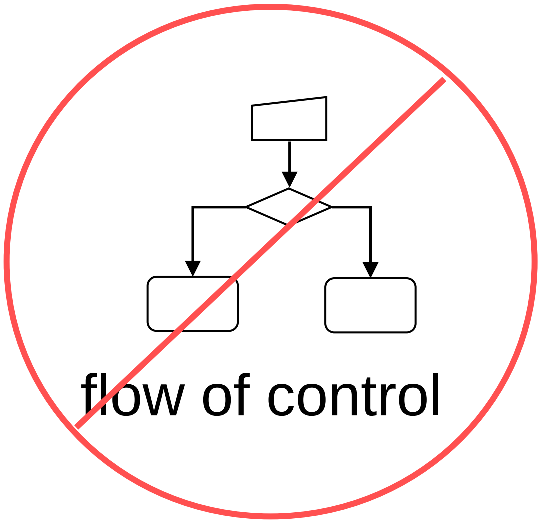
end-spec

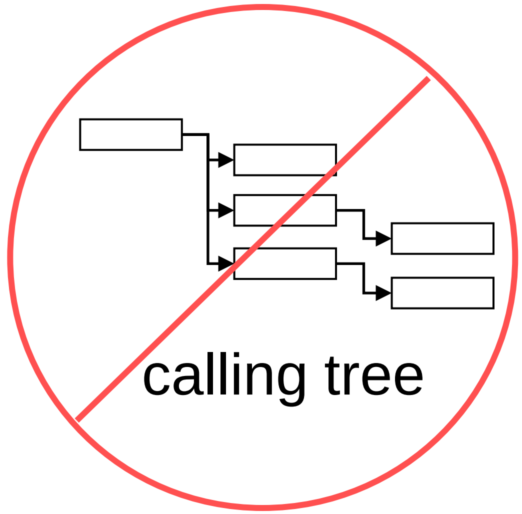
Main spec activity: composition and refinement

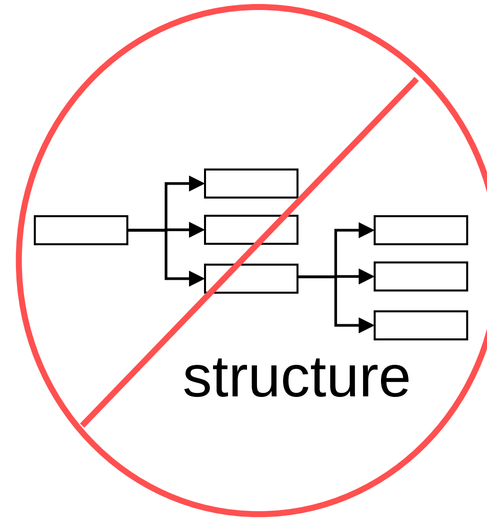


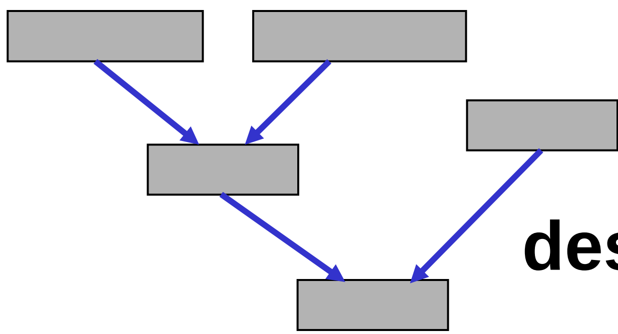
Kestrel
Institute

Specware diagrams have *design* arrows









design arrows

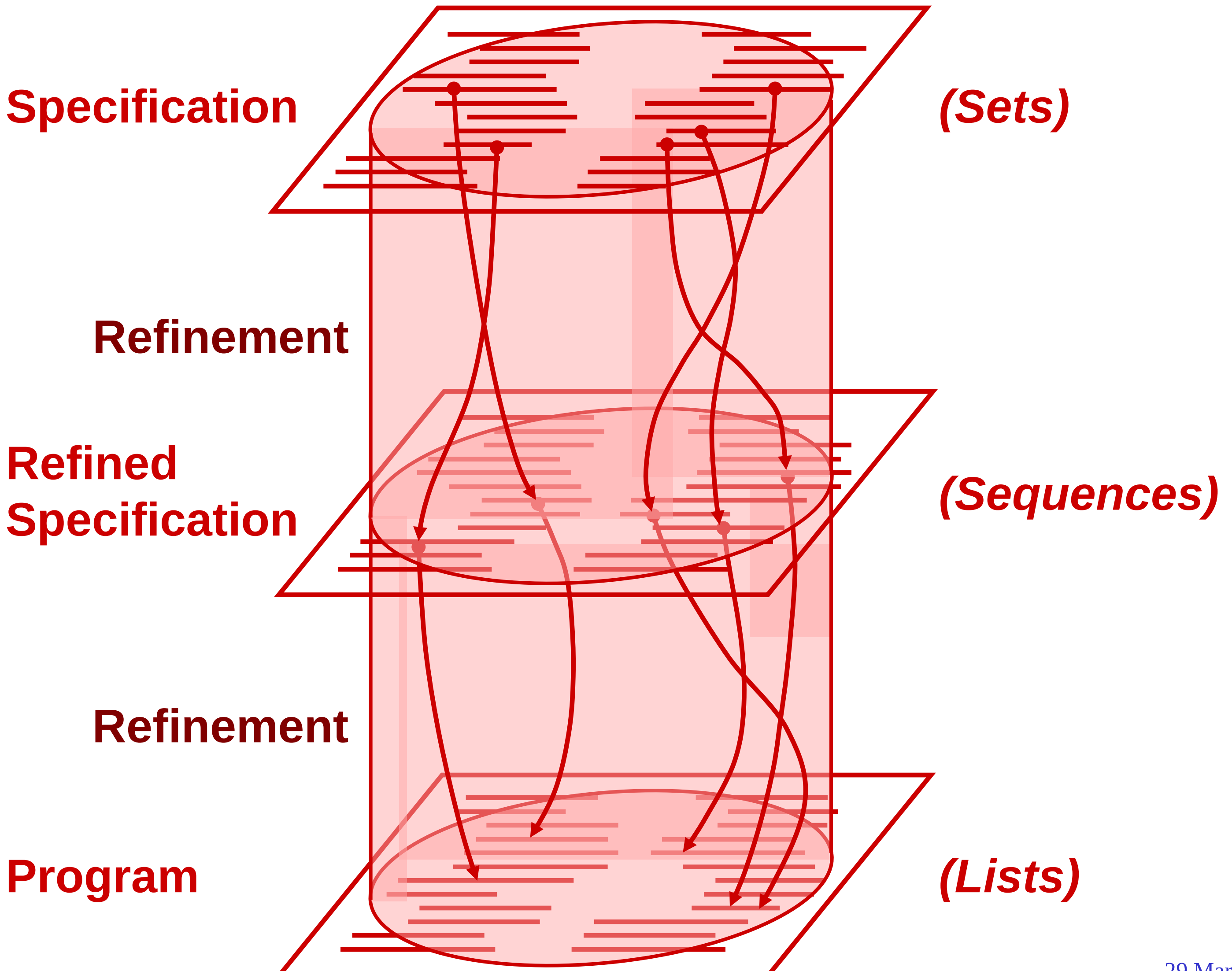
Composition & refinement with semantics = Design → You reach executability via design.

Benefits

- Keep concepts isolated
  - Smaller, clearer, more tractable
- Efficient composition
  - Common elements unified
- Separation
  - “what” from “how”
  - “how” can be implemented in stages (stepwise refinement)
- Assurance
  - Proof obligations identified

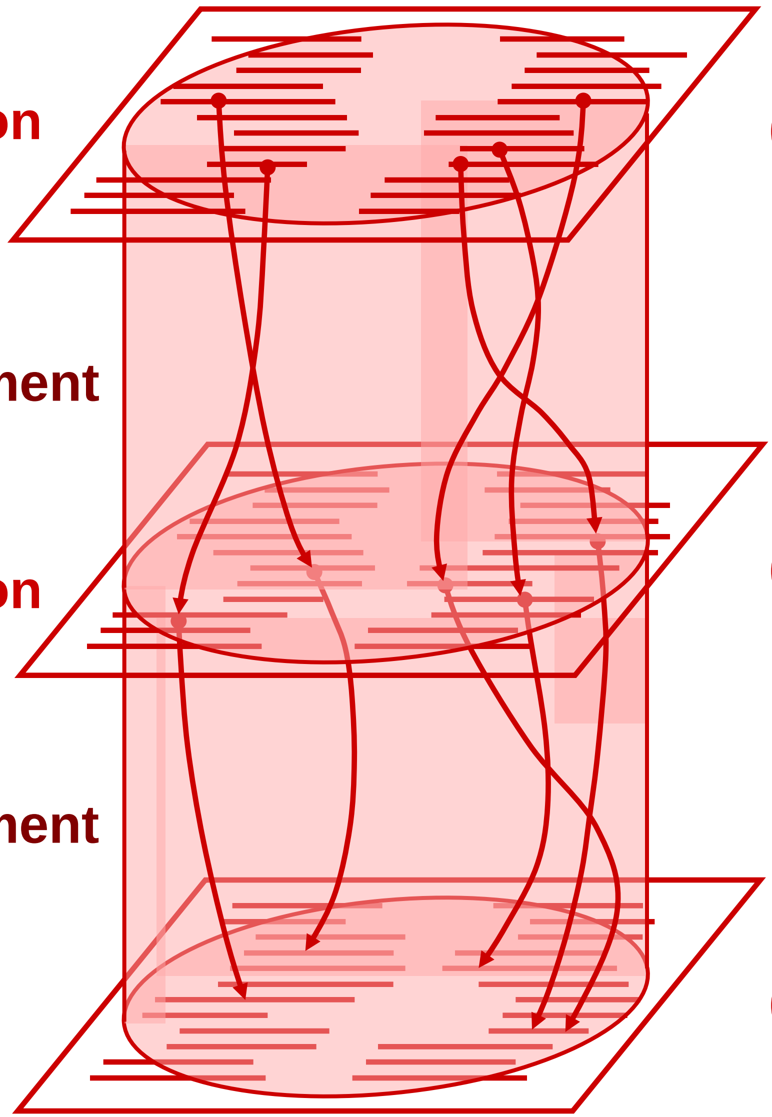
Kestrel
Institute

Refinement example: Sets



Kestrel
Institute

Refinement example: Passwords



Specification *(Passwords)*

Refinement

Refined Specification *(Naturals)*

Refinement

Program *(Integers)*

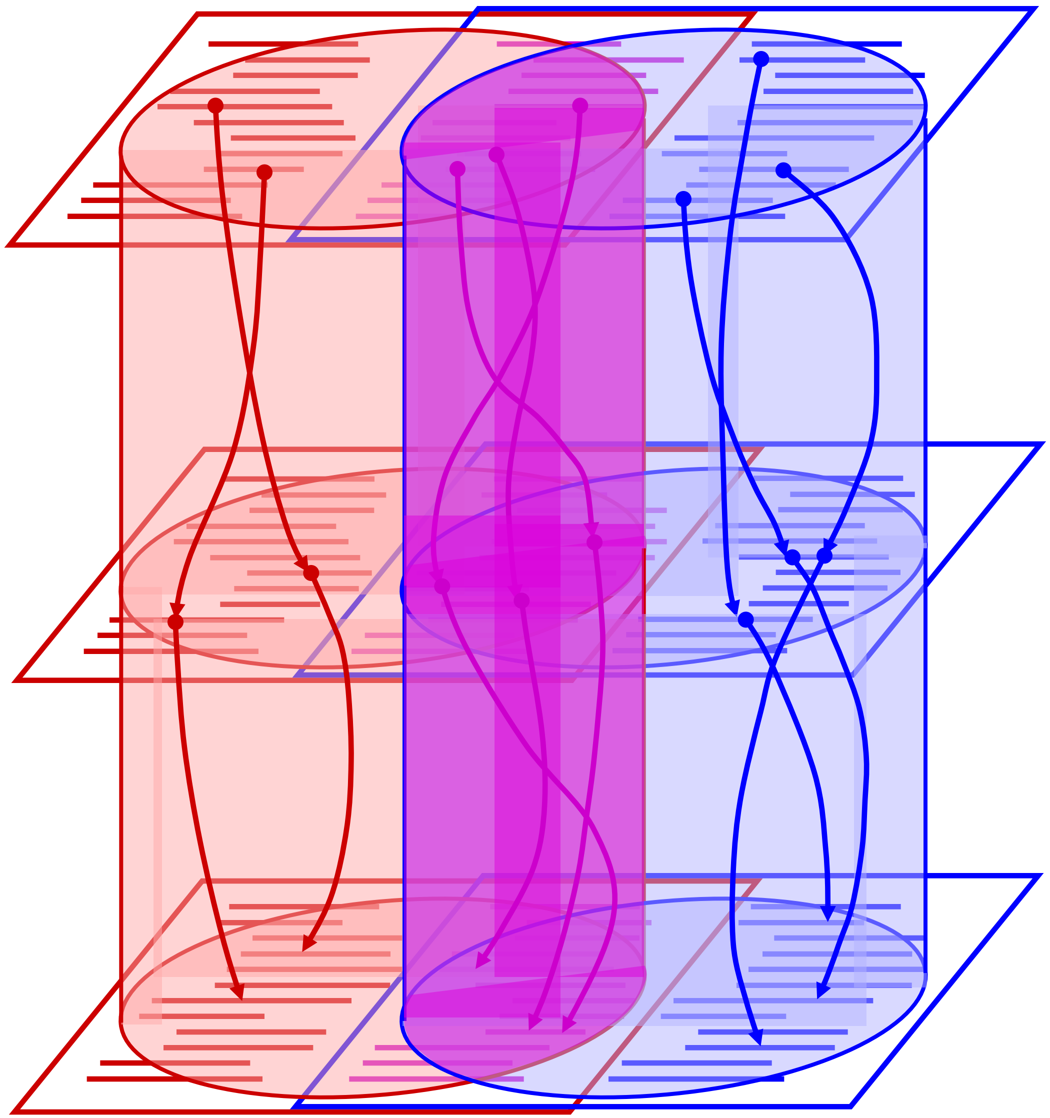
Kestrel Institute

Refinement of a composition

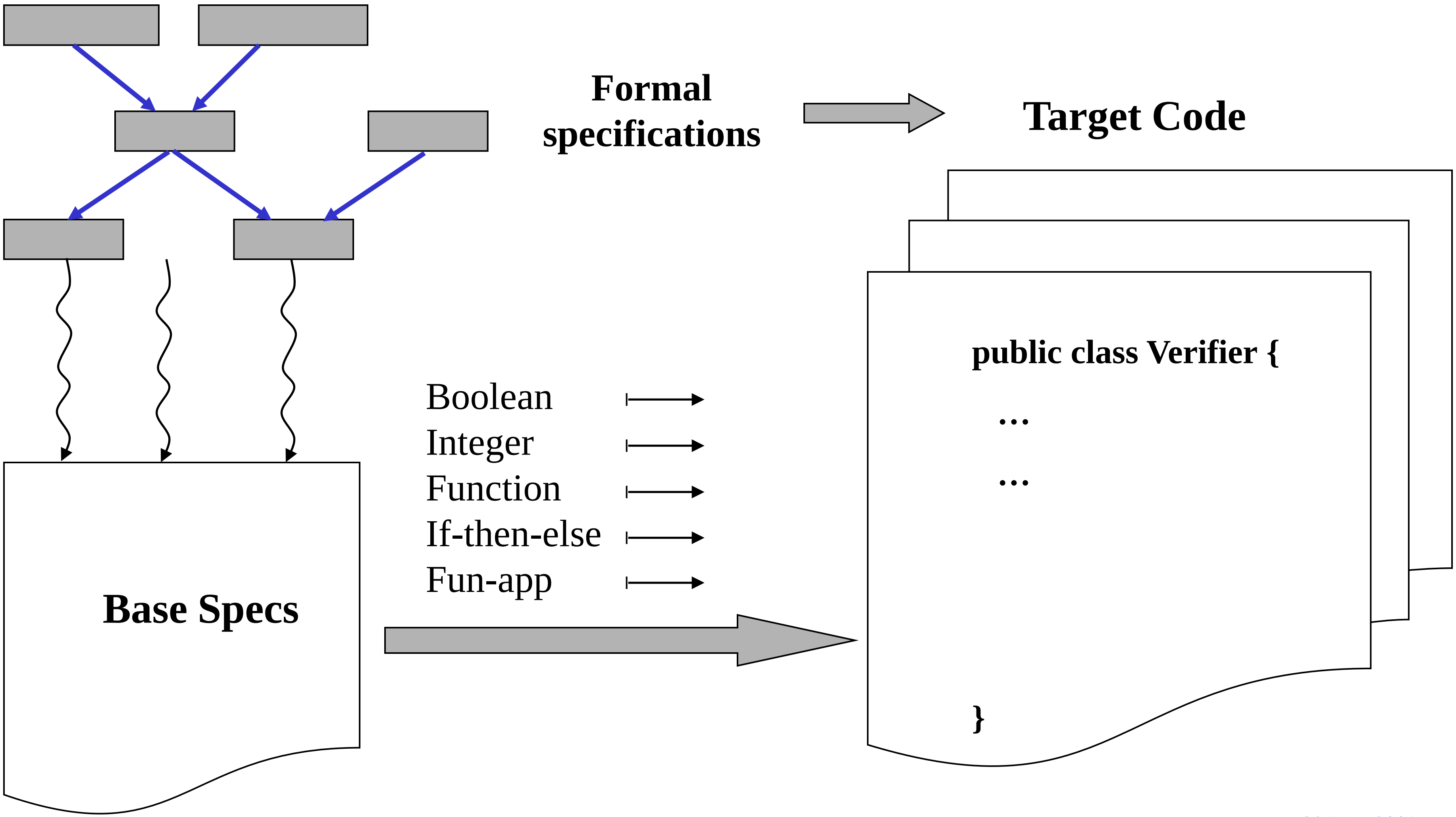
Sets of Passwords

Sequences of Naturals

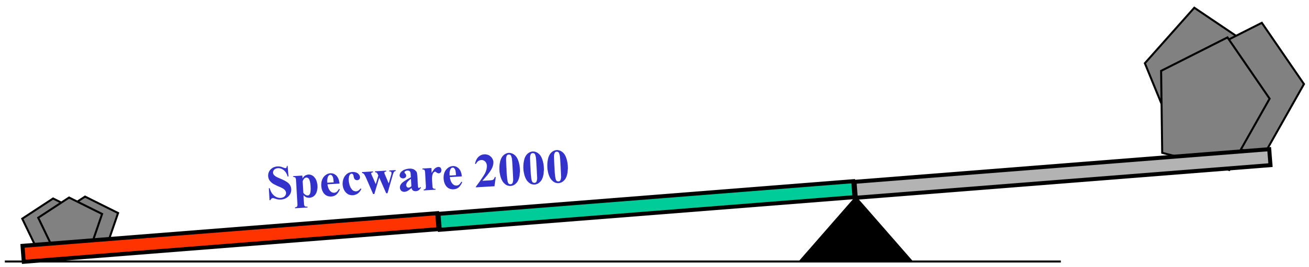
Lists of Integers



Code Synthesis



Source of Specware's leverage



Sound Mathematical Foundation

Composition & refinement engine

Categories	Diagrams of diagrams	Logics
Morphisms	Grothendieck construction	p-specs
Colimits	Natural transformations	…and more

Software Design Knowledge

Libraries of optimizations, tactics, design strategies

Divide & conquer	Context dependent simplification
Partial evaluation	Global search
Finite differencing	…and more

Using Specware's leverage

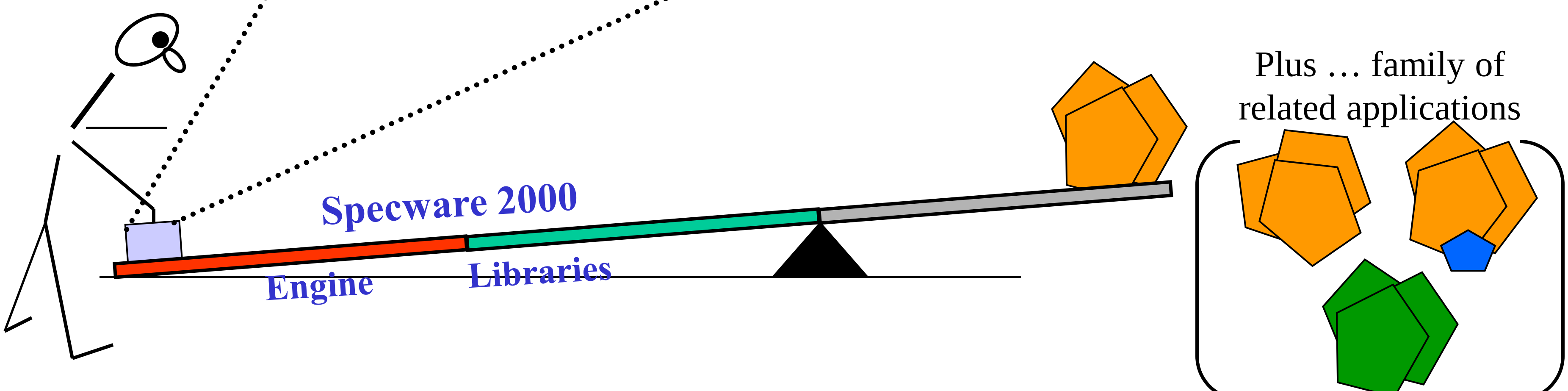
Application Domain Knowledge

- •Theory #1
- •Theory #2
- .
- .
- •Theory #T

⇨

Payoff:

- –Assurance
- –Maintainability
- –Reuse
- –Performance



Kestrel Institute

From research to operations

Generic algorithms & system design lead to…

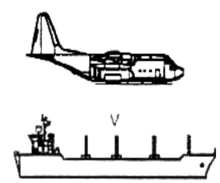
Research Prototype

Software Expert (Ph.D.)

e.g. "divide-and-conquer design knowledge"

a scheduler generator which produces an …

Domain Models



Application Expert (Ph.D.)

add e.g. ship model

operational scheduler

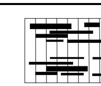
Policy Requirements

location-specific schedule generator FAA, DOD

Policy Analysts (M.S., C.S.)

e.g. crew scheduling

Operation Requirements



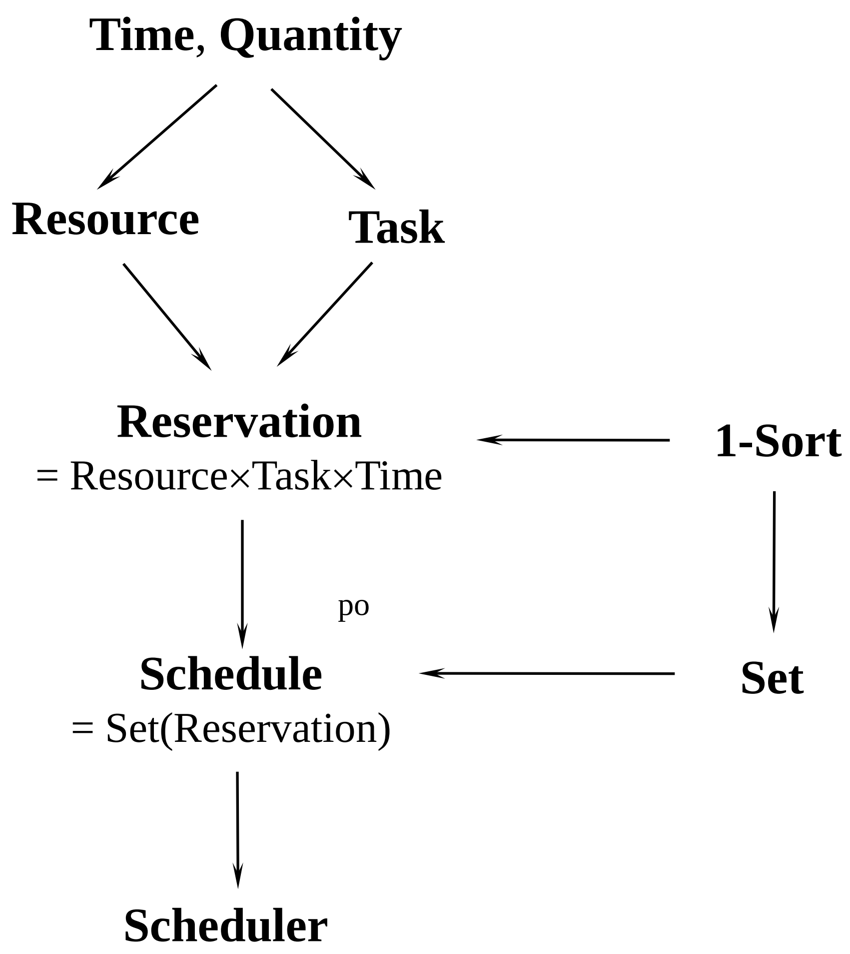
operation schedule

Operations Manager

caption

level of knowledge required

A structured spec for scheduling



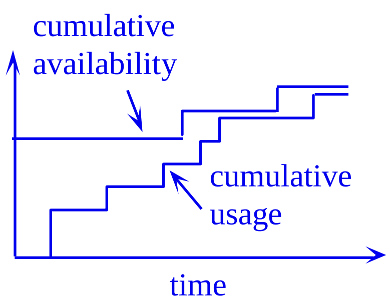
Transportation domain modeling

Resource

Consumable

examples: fuel, crew time

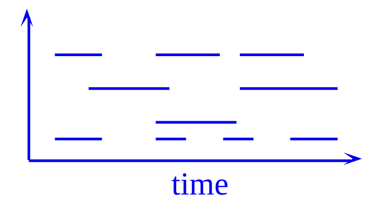
constraint: cum. use ≤ cum. avail



Reusable

examples: parking lots, ramp space, parallel processors, power

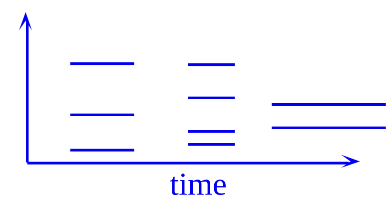
constraints: upper bound on capacity
finite usage intervals



Synchronously Reusable

examples: ship, aircraft, truck

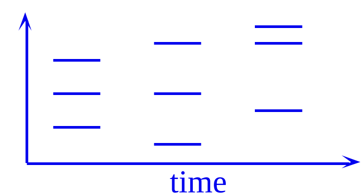
constraints: synchronized blocks of reservations
min separation between blocks



Exact Capacity

example: wafer oven

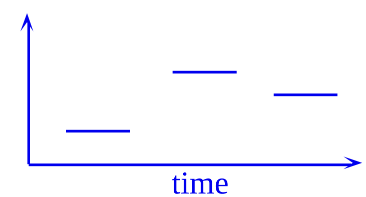
constraint: lb = ub on capacity



Nonsharable

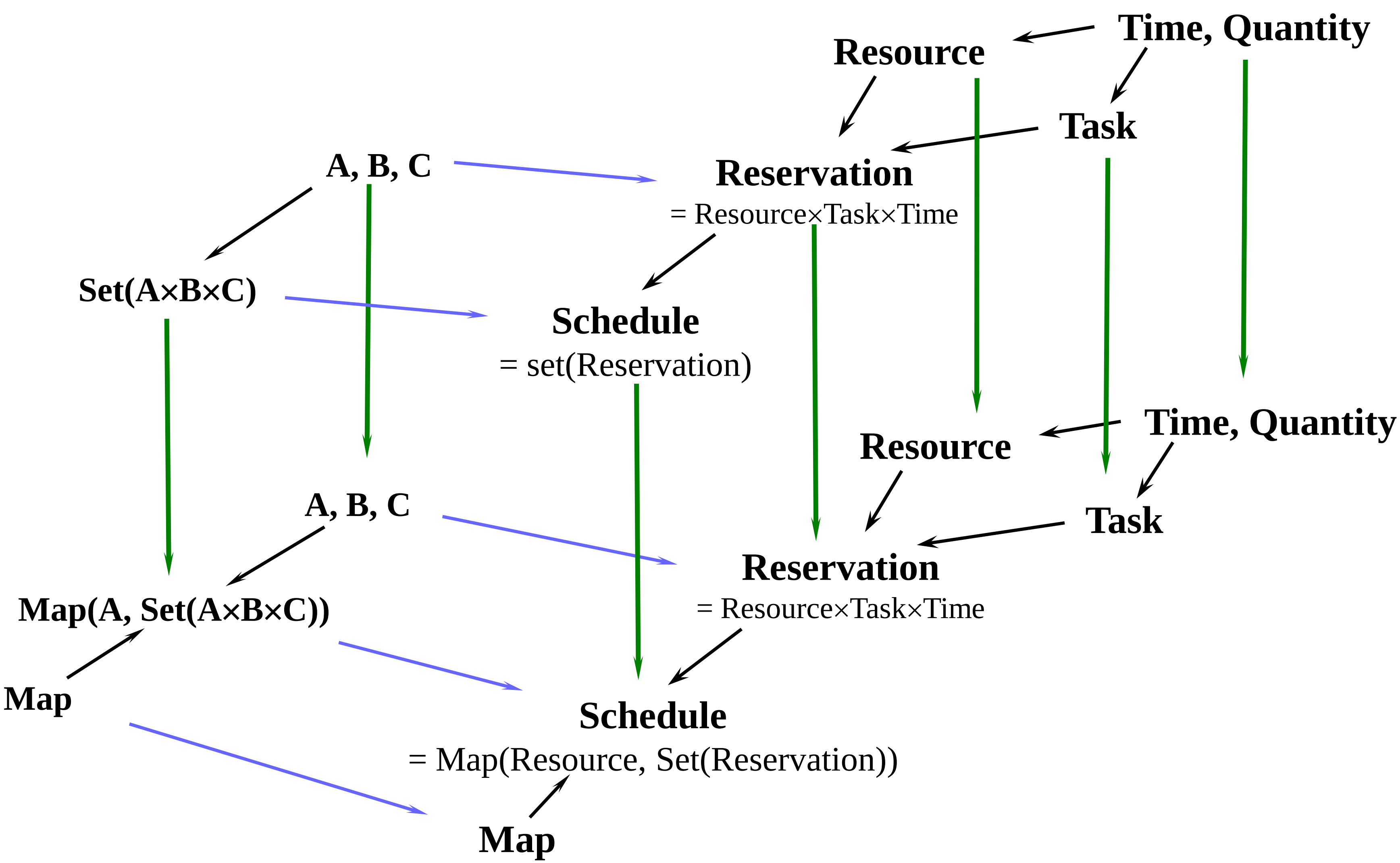
examples: berth, runway, crew

constraint: capacity = 1



Kestrel
Institute

Diagram Refinement



Planware Generator

Resource

Parameter	Lower Bound	Exact Value	Upper Bound
Start Time	*Task.release*		*Task.pick-up*
Resource-type		*Multi-choice menu*	*Sum of task req'd resources*
Instantaneous Demand		*Sum of task demands*	

Kestrel Institute

Planware Generator–2

Resource → *Reusable resource*

Parameter	Lower Bound	Exact Value	Upper Bound
Start Time	*Task.release*	*Finish - Dur*	*Task.pick-up*
Resource-type		*Multi-choice menu*	*Sum of task req'd resources*
Instantaneous Demand	*min-cap*	*Sum of task demands*	*max-cap*
Duration	*0*	*Finish – Start*	
Finish Time	*Task.ead*	*Start + Dur*	*Task.due-date*
Max-capacity		*r.r-type.max-cap*	

Also: *Precedes, Min-capacity*

Planware Generator–3

Resource

↓

Reusable resource

↓

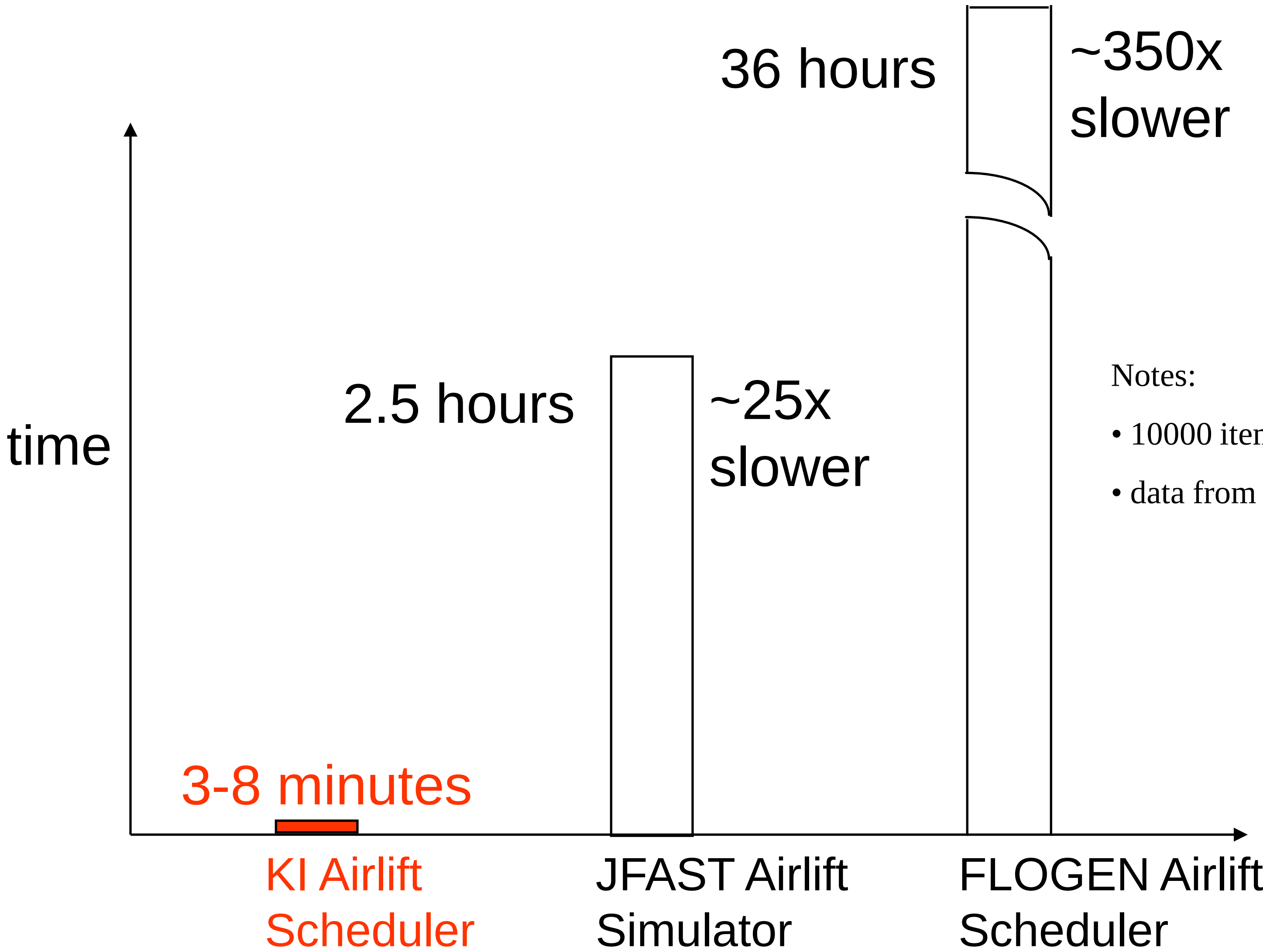
Synchronous resource

Parameter	Lower Bound	Exact Value	Upper Bound
Start Time	*Task.release*	*Finish - Dur*	*Task.pick-up*
Resource-type		*Multi-choice menu*	*Sum of task req'd resources*
Instantaneous Demand	*min-cap*	*Sum of task demands*	*max-cap*
Duration	*0*	*Finish – Start*	
Finish Time	*Task.ead*	*Start + Dur*	*Task.due-date*
Max-capacity		*r.r-type.max-cap*	
Separation	*0*	*r.r-type.separation*	

Also: *Precedes, Min-capacity*

Kestrel Institute

Performance



Notes:

- 10000 item movements
- data from 1997

Important users

- Motorola
- NSA
- Boeing
- Kestrel Institute (KI)
- Kestrel Technology LLC (KT)
- Other (Georgia Tech, Lockheed Martin, …)

Motorola work

- Peter White, Conan Dailey, et al.
- Used Specware 1.x to create a specification for an OS separation kernel
- Successful application
  - Security proven to NSA
  - Embedded in commercially available AIM processor

Kestrel Institute

NSA experiment

Bake-Off : Two teams given

- same requirements document
- same time
- same funds

Each team implemented the system independently, and a third party tested code and awarded reliability scores.

Methodologies Used

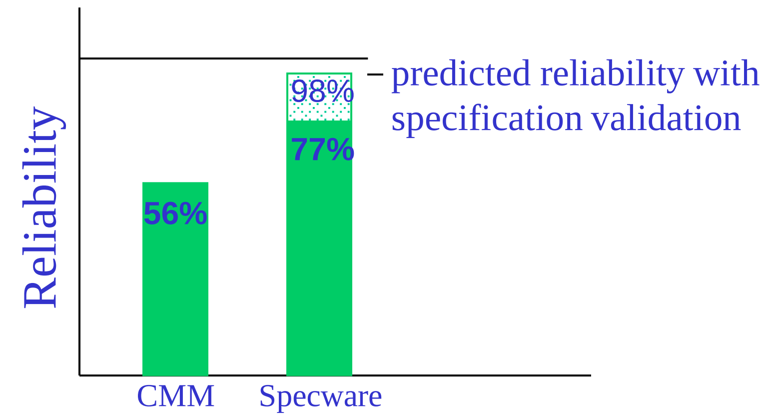
Specware

Specware specification & code synthesis

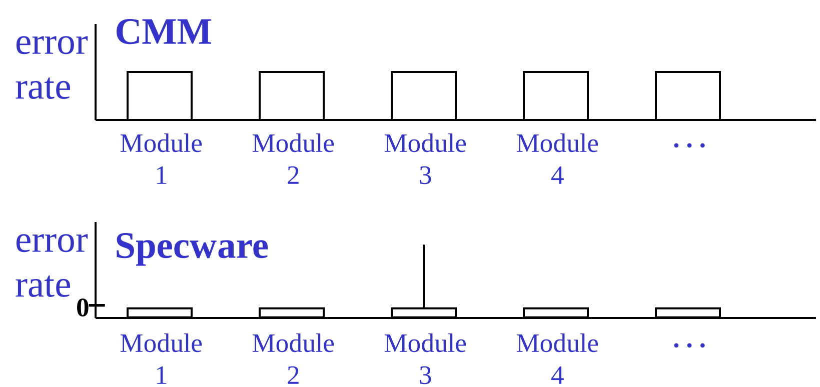
CMM

Software Engineering Institute Capability Maturity Model Level 4 with UML specification & initial design

Reliability Scores for Critical Functionality



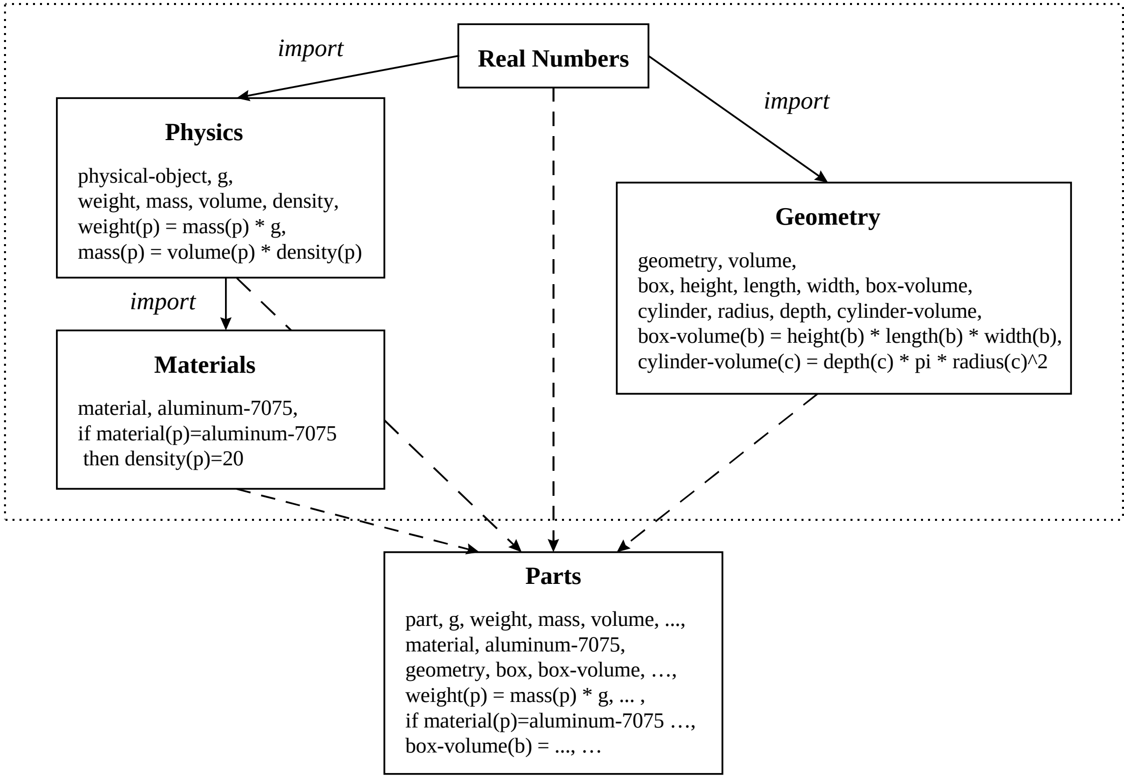
Distribution of Code Errors



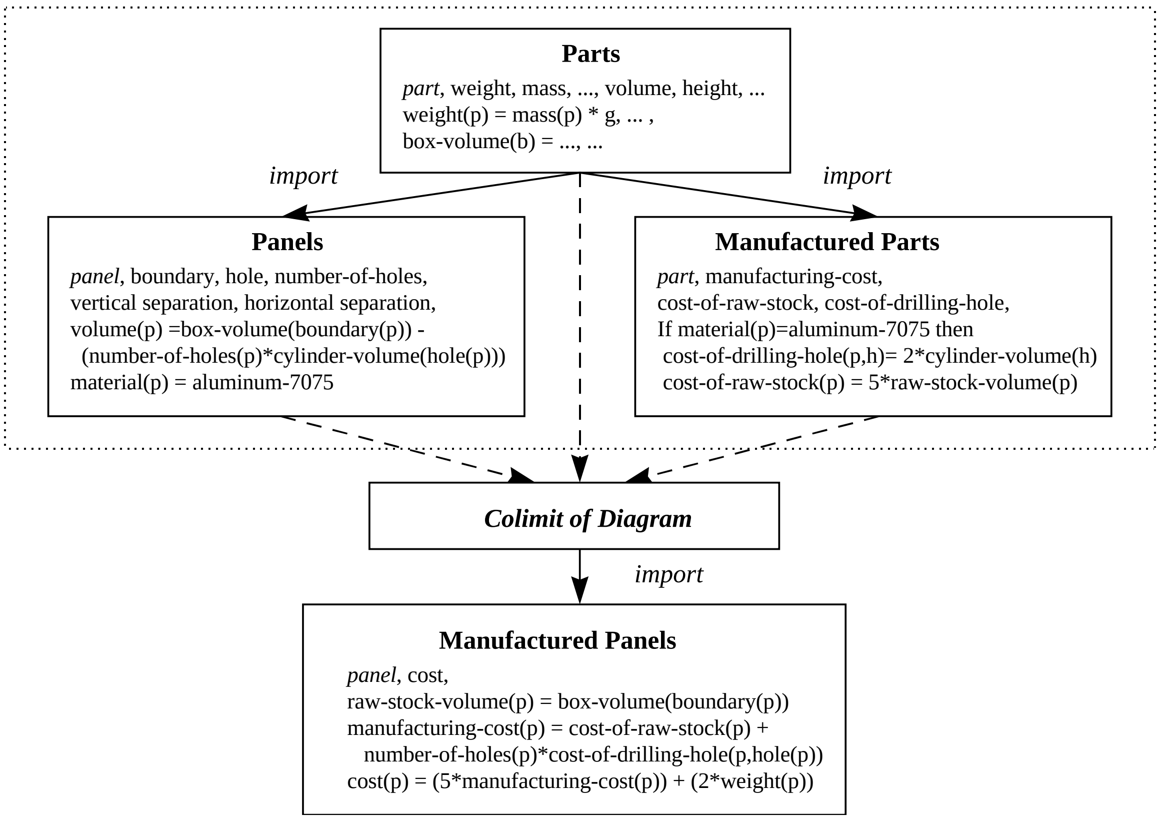
Kestrel Institute

Boeing work

- FAA-compliant electronic equipment rack layout
  - Maintain separations
  - Maintain redundancy
  - Maintain ease of access
  - Minimize costs
    - Cable length, etc.
  - Etc., etc.

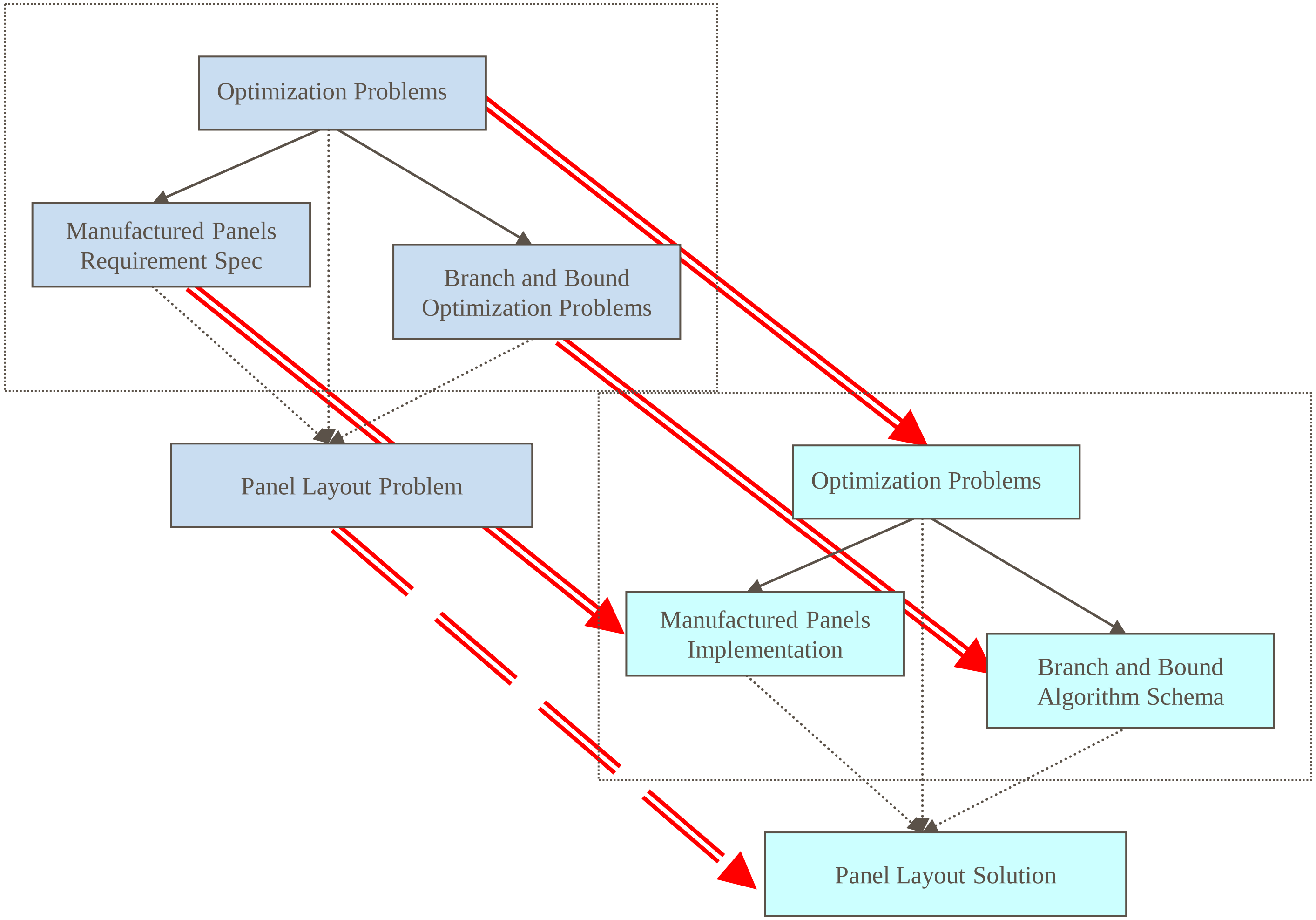


Boeing (cont’d)



Kestrel Institute

Boeing (cont'd)



Kestrel's recent work using Specware

- High assurance Java virtual machine
- Network vulnerability analysis
- Bootstrapping Specware in its own language

Kestrel Institute

Java bytecode verifier

- First complete formal executable specification and validation of bytecode verification
- Uncovered several flaws in the informal specification
- Designed & tested specification modifications to eliminate those flaws and enhance the performance
- By-product: reusable components, e.g, a data flow analysis engine

Described later in this workshop

Network vulnerability analysis

- Detects vulnerabilities in COTS software applications
- Works on byte code → usable even when sources aren't available
- Early stages of work

Described later in this workshop

Kestrel Institute

What's new in Specware?

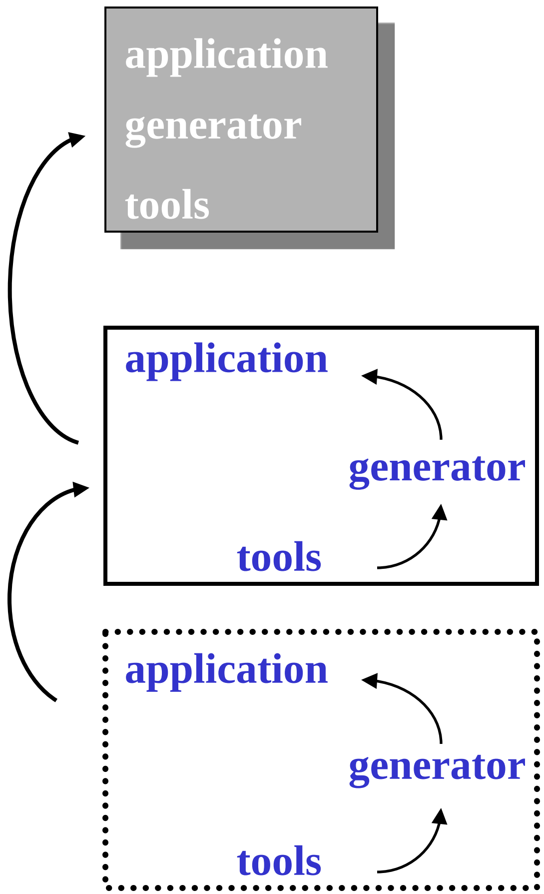
- Language features
  - 1st order polymorphism
  - automatic relax/restrict
  - imperative constructs
- Much more compact & readable syntax
  - record notation
  - named co-products
  - infix operations
- Bootstrapped system
- Larger test suite (including Specware)
- Meta-language for programmable tactics & replay
- Prover Integration {Snark, Gandalf, …}
- Hosting on Wintel & Linux
- Refine-free (simpler licensing)

Roles of KI and KT

NewCo: COTS

KT: tech transition

KI: basic research



- **Kestrel Institute (KI)**
  - Non-profit R&D
  - Emphasis on basic and exploratory research
  - Contained growth
  - Core technology feeding diversity of applications
  - Academic spirit
- **Kestrel Technology LLC (KT)**
  - For-profit R&D
  - Emphasis on service for using and extending KI technology
  - Growth-oriented
  - Narrow application focus
  - Commercial spirit
  - Spin-out companies

BACKUP SLIDES AFTER HERE

Development plans

- Ongoing work
  - Hereditary diagrams
  - Optimizations
  - C code synthesis
- Language extensions
  - Dependent types
  - Relax constraint on morphisms and sort-structure
  - Non-deterministic operators
- Inference
  - Extend Gandalf
  - Extend inference tactics
- Spec categories
  - Support for theory slicing
  - Support for targeting imperative and OO languages
- High performance output code
  - C, Java
- Designware
  - Application support libraries
  - Move Slang-based libraries into MetaSlang
- Java-based GUI
  - Interface to diagrams
  - Interface to Designware

Example of a colimit

spec BINARY-RELATION is
  sort E
  op $\_br\_ : E, E \rightarrow Boolean$
end-spec

spec REFLEXIVE-RELATION is
  sort E
  op $\_rr\_ : E, E \rightarrow Boolean$
  axiom reflexivity is $a\ rr\ a$
end-spec

spec TRANSITIVE -RELATION is
  sort E
  op $\_tr\_ : E, E \rightarrow$ Boolean
  axiom transitivity is
    $a\ tr\ b \wedge b\ tr\ c \Rightarrow a\ tr\ c$
end-spec

spec PREORDER-RELATION is
  sort E
  op $\leq : E, E \rightarrow Boolean$
  axiom reflexivity is
    $a \leq a$
  axiom transitivity is
    $a \leq b \wedge b \leq c \Rightarrow a \leq c$
end-spec

World class research

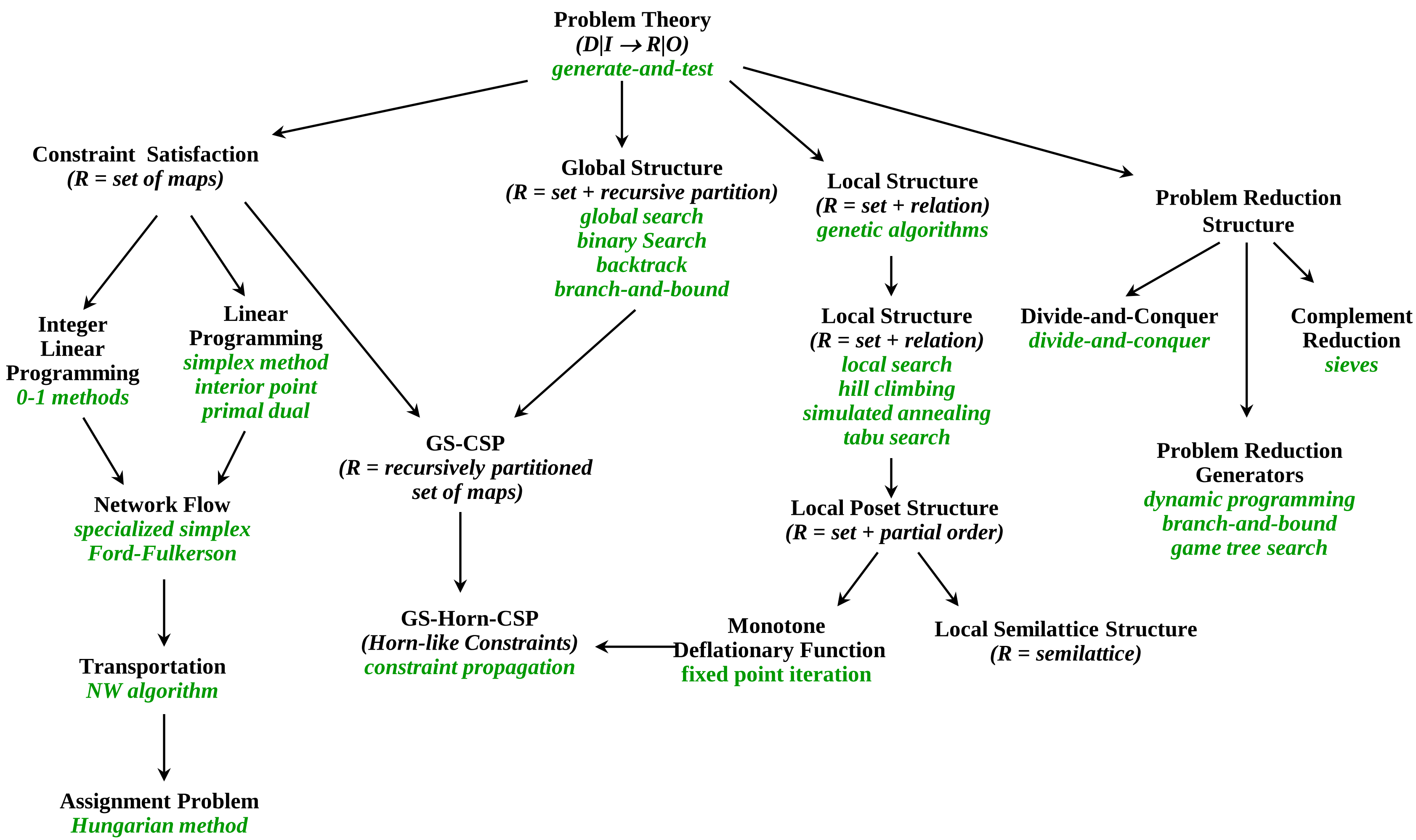
- Director
  - Fellow of the ACM
  - Winner of the Grace Hopper Award
  - Consultant to the Defense Science Board
  - Adjunct professor at Stanford
- CTO
  - Fellow of the AAAI
  - Former chair of IFIPS 2.1
  - Adjunct professor at Stanford
- Staff includes:
  - Current chair of IFIP 2.1
  - Several DARPA PIs
  - Experts in
    - Category theory
    - Program synthesis
    - Functional programming
    - Java security
    - Optimization
    - Algorithm design and synthesis
    - Resource allocation
    - Network optimization
    - Signal processing
    - …and more

FAQs

1. Do you have an independent proof of correctness of generated code?
2. Do you think I would write in MetaSlang?
3. Do you expect me to maintain MetaSlang?
4. Why not just use C++, Java, Haskell, B, PVS, …?
5. What about my existing body of code?
6. Can your output code outperform my hand-crafted code?

Kestrel Institute

Taxonomy of algorithm theories



Basic operations specs

```
spec BIN-OP is
  sort U
  op f : U * U -> U
  end-spec
```

```
spec COMMUTATIVE-BIN-OP is
  import BIN-OP
  axiom commutativity is  fa(x,y) f(x,y) =
      f(y,x)
  end-spec
```

```
spec IDEMPOTENT-BIN-OP is
  import BIN-OP
  axiom idempotence is  fa(x) f(x,x) = x
  end-spec
```

```
spec ASSOCIATIVE-BIN-OP is
  import BIN-OP
  axiom associativity is  fa(x,y,z) f(x,f(y,z)) =
      f(f(x,y),z)
  end-spec
```

```
spec BIN-OP-w-ID is
  import BIN-OP
  op id : U
  axiom left-identity is   fa(x) f(id,x) = x
  axiom right-identity is  fa(x) f(x,id) = x
  end-spec
```

```
spec BIN-OP-w-ABS is
  import BIN-OP
  op abs : U
  axiom left-absorption is   fa(x) f(abs,x) = abs
  axiom right-absorption is  fa(x) f(x,abs) = abs
  end-spec
```

Kestrel Institute

Semilattice

```
def SEMILATTICE-import : Spec =
 diagramColimit("SEMILATTICE-import",
          [BIN-OP,
           COMMUTATIVE-BIN-OP,
           IDEMPOTENT-BIN-OP,
           ASSOCIATIVE-BIN-OP],
          [BIN-OP !--> COMMUTATIVE-BIN-OP,
           BIN-OP !--> IDEMPOTENT-BIN-OP,
           BIN-OP !--> ASSOCIATIVE-BIN-OP])
```

```
spec SEMILATTICE is

 import SEMILATTICE-import

 op pord1 : U * U -> Boolean
 def pord1(x,y) = (f(x,y) = x)

 op pord2 : U * U -> Boolean
 def pord2(x,y) = (f(x,y) = y)

 end-spec
```

Semilattice

```
def SEMILATTICE-w-ID : Spec =
 diagramColimit("SEMILATTICE-w-ID",
            [BIN-OP,
             SEMILATTICE,
             BIN-OP-w-ID],
            [BIN-OP !--> SEMILATTICE,
             BIN-OP !--> BIN-OP-w-ID])
```

```
def SEMILATTICE-w-ABS : Spec =
 diagramColimit("SEMILATTICE-w-ABS",
            [BIN-OP,
             SEMILATTICE,
             BIN-OP-w-ABS],
            [BIN-OP !--> SEMILATTICE,
             BIN-OP !--> BIN-OP-w-ABS])
```

```
def SEMILATTICE-w-ID-n-ABS : Spec =
 diagramColimit("SEMILATTICE-w-ID-n-ABS",
            [SEMILATTICE,
             SEMILATTICE-w-ID,
             SEMILATTICE-w-ABS],
            [SEMILATTICE !--> SEMILATTICE-w-ID,
             SEMILATTICE !--> SEMILATTICE-w-ABS])
```

```
def BV-DATA-FLOW : Spec =
 diagramColimit("BV-DATA-FLOW",
        [DATA-FLOW-param,
         DATA-FLOW,
         TRANSFER-FUNCTIONS,
         MAPS],
        [DATA-FLOW-param !-->
    DATA-FLOW,
         DATA-FLOW-param --->
    TRANSFER-FUNCTIONS
         where ["U"     |-> "BVSL",
                "f"     |-> "join",
                "id"    |-> "btm",
                "abs"   |-> "top",
                "pord1" |-> "gtq",
                "pord2" |-> "ltq"],
         MAPS !--> DATA-FLOW,
         MAPS !--> TRANSFER-
    FUNCTIONS])
```