Stream processing Spade

Opening the Levees for Stream Processing

IBM PL Day 7 May 2009

Martin Hirzel, Henrique Andrade, Buğra Gedik, Vibhore Kumar, Giuliano Losa, Robert Soulé, Kun-Lung Wu

About this Talk

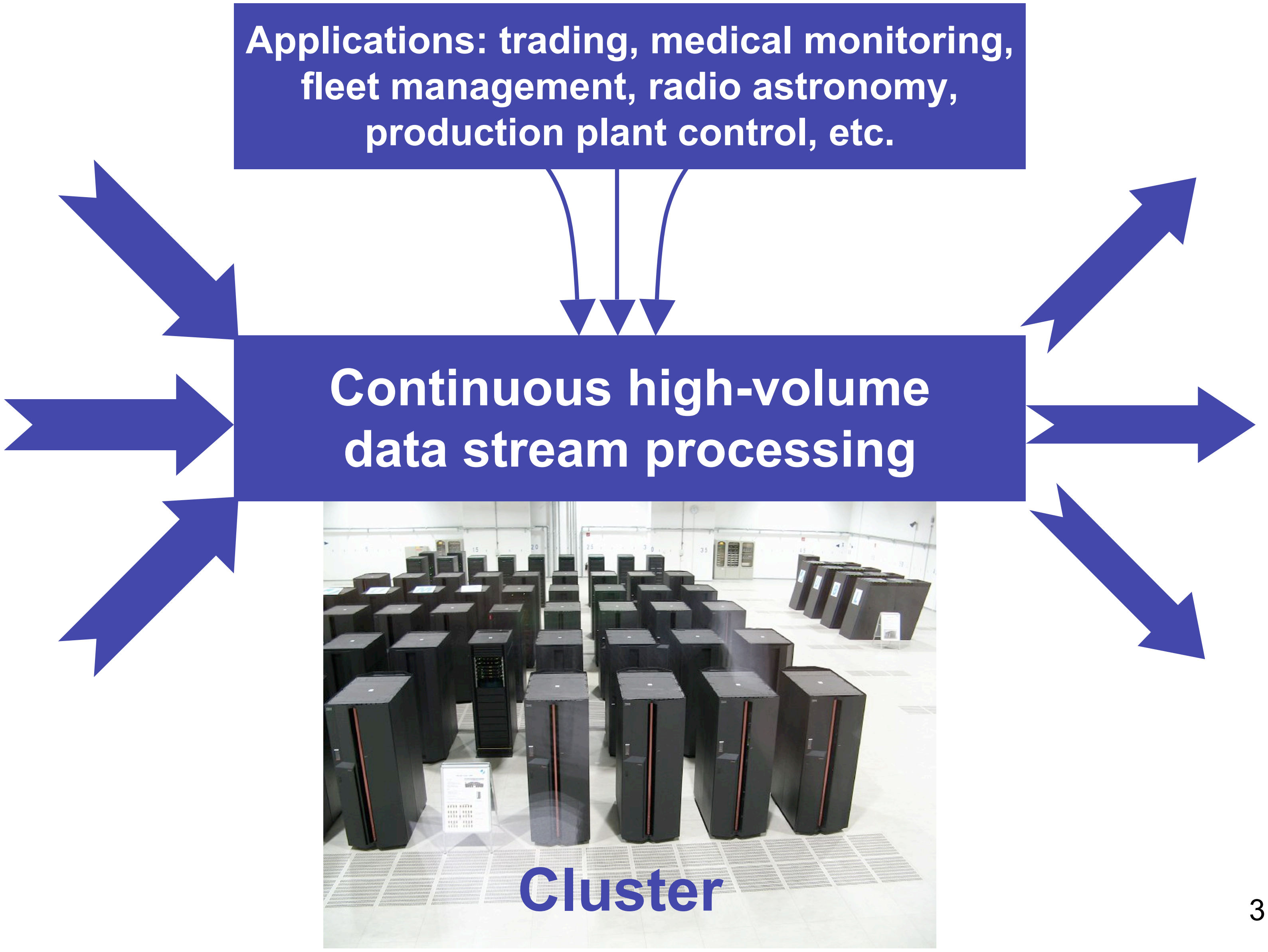
Technical

- A language for stream computing (work in progress)

Non-technical

- Exercise: pragmatic PL design
- Experience: PL guy in different domain

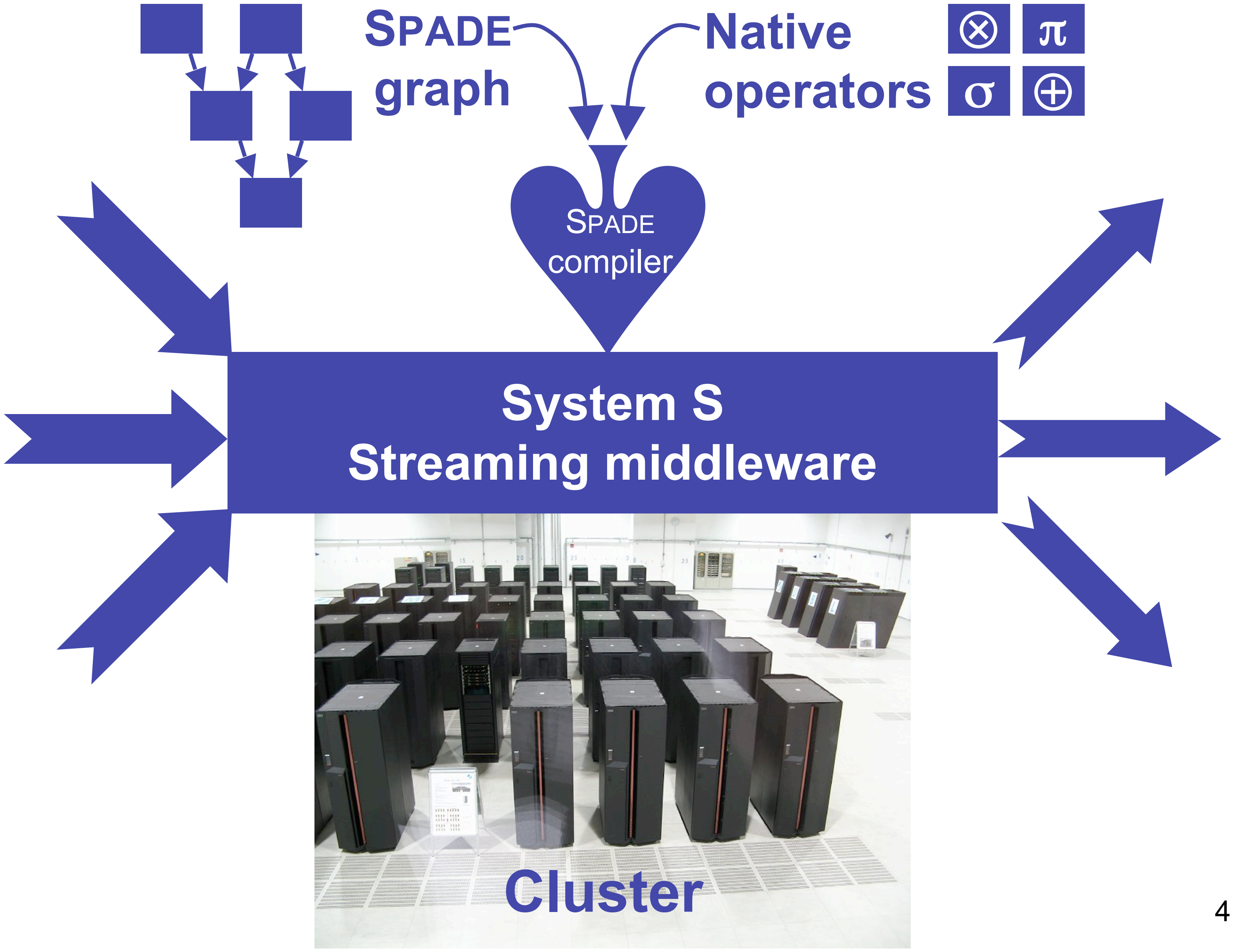
Cluster Stream Processing



Applications: trading, medical monitoring, fleet management, radio astronomy, production plant control, etc.

Continuous high-volume data stream processing

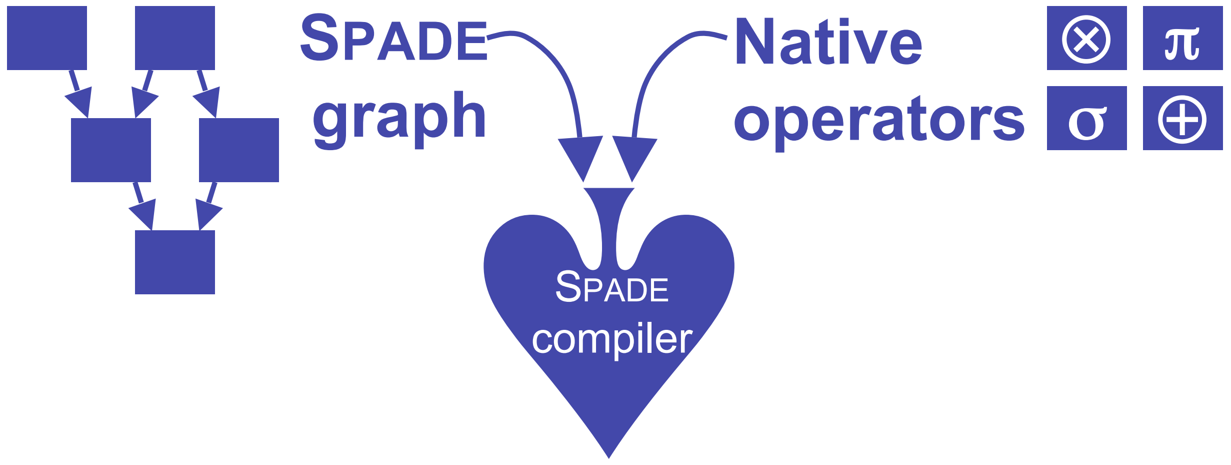
SPADE and System S



Outline

- Systems Solution
- **Language Problem**
- Language Design
- Design Process
- Future Work

PL Problem



Problem:

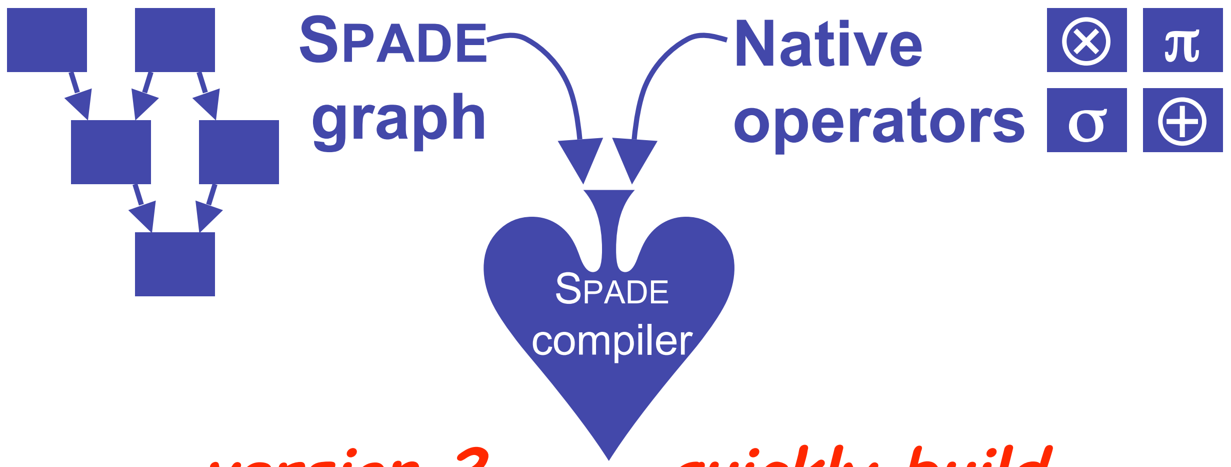
Design SPADE language and compiler.

Priorities:

Performance:	Ultra-fast streaming on a cluster.
Generality:	Support diverse set of applications.
Usability:	Hide complexity of distributed systems.

PL Problem

technical and nontechnical



Problem:

version 2 quickly build

Co-Design SPADE language and compiler.

Priorities:

with System S

Performance:	Ultra-fast streaming on a cluster.
Generality:	Support diverse set of applications.
Usability:	Hide complexity of distributed systems.

beyond StreamIt, StreamSQL, …

also, interface with (new and legacy) native code

Outline

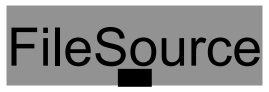
- Systems Solution
- Language Problem
- Language Design
- Design Process
- Future Work

Terminology

Stream

- Infinite sequence of tuples
- Edge in stream graph

Operator



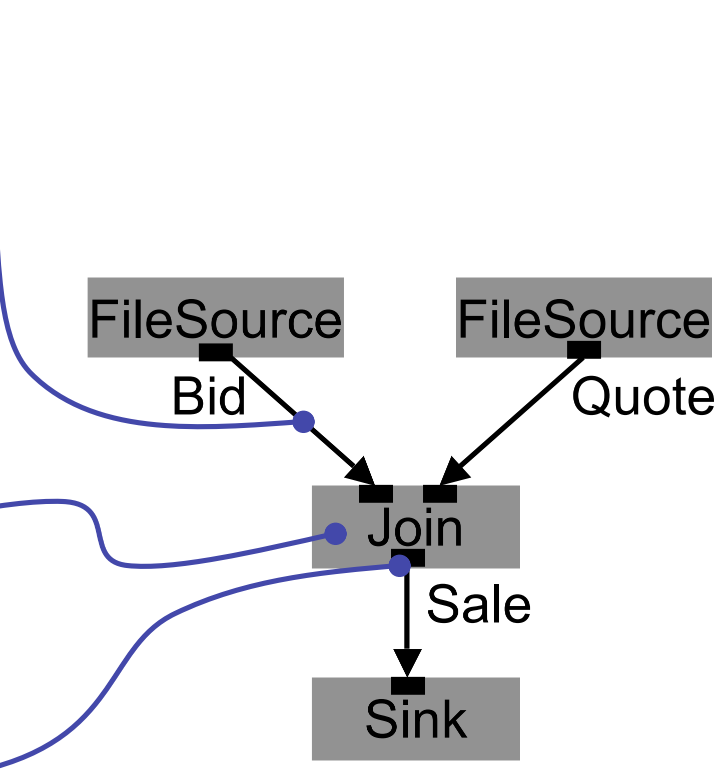
- Reusable stream transformer
- May be primitive or composite

Operator invocation

- Defines its output streams
- Vertex in stream graph

Port

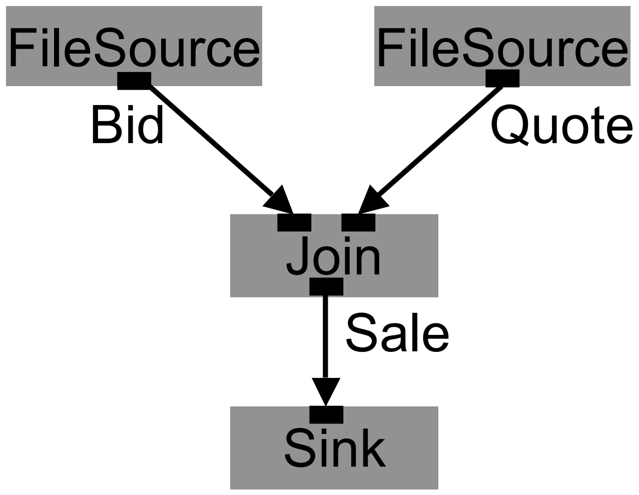
- Point where streams connect to operator



SPADE Stream Graph

```
stream<…> Bid = FileSource() {              //1
  …                                         //2
}                                           //3
stream<…> Quote = FileSource() {            //4
  …                                         //5
}           Operator   Port                 //6
stream<…> Sale = Join(Bid; Quote) {         //7
  …       Stream                            //8
  …                                         //9
  …                                         //10
  …                                         //11
}                                           //12
() = FileSink(Sale) { … }                   //13
```

Operator invocation



SPADE Types

```
stream<string buyer, string item, decimal64 price> Bid = FileSource() {          //1
  …                                                                              //2
}                                                                                //3
stream<string seller, string item, decimal64 price> Quote = FileSource() {       //4
  …                                                                              //5
}                                                                                //6
stream<string buyer, string seller, string item> Sale = Join(Bid; Quote) {       //7
  …                                                                              //8
  …                                                                              //9
  …                                                                              //10
  …                                                                              //11
}                                                                                //12
() = FileSink(Sale) { … }                                                        //13
```

SPADE Operator Customization

```
stream<string buyer, string item, decimal64 price> Bid = FileSource() {          //1
  param    fileName  : "BidSource.dat"; format: csv;                              //2
}                                                                                 //3
stream<string seller, string item, decimal64 price> Quote = FileSource() {       //4
  param    fileName  : "SaleSource.dat"; format: csv;                             //5
}                                                                                 //6
stream<string buyer, string seller, string item> Sale = Join(Bid; Quote) {       //7
  window   Bid       : sliding, time(30);                                         //8
           Quote     : sliding, count(50);                                        //9
  param    match     : Bid.item == Quote.item && Bid.price >= Quote.price;        //10
  output   Sale      : item = Bid.item;                                           //11
}                                                                                 //12
() = FileSink(Sale) { param fileName: "Result.dat"; format: csv; }                //13
```

SPADE Operator Definition

- Previous slides invoke and customize operators, but don't define them.
- Support for 2 kinds of operator definition

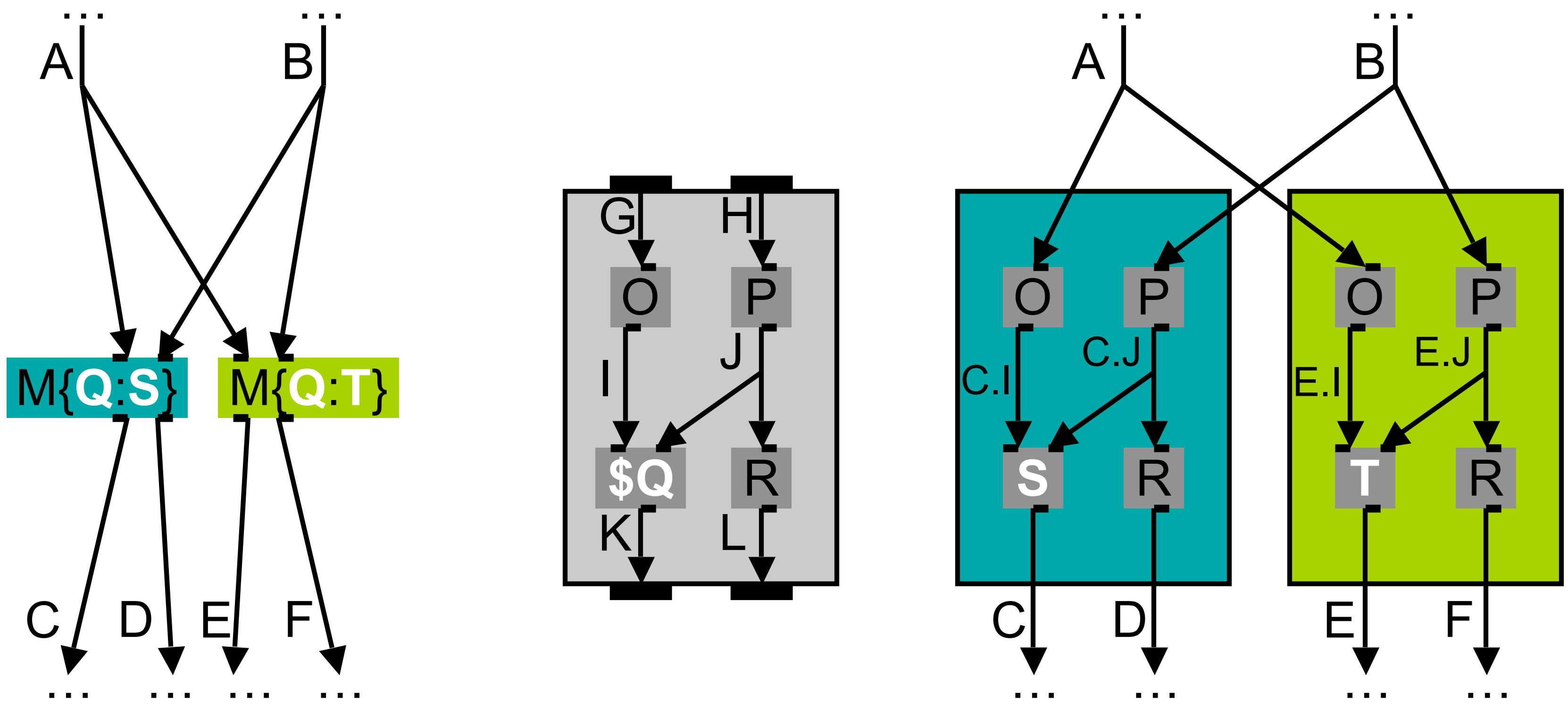
SPADE Operator Definition

- Previous slides invoke and customize operators, but don't define them.
- Support for 2 kinds of operator definition:

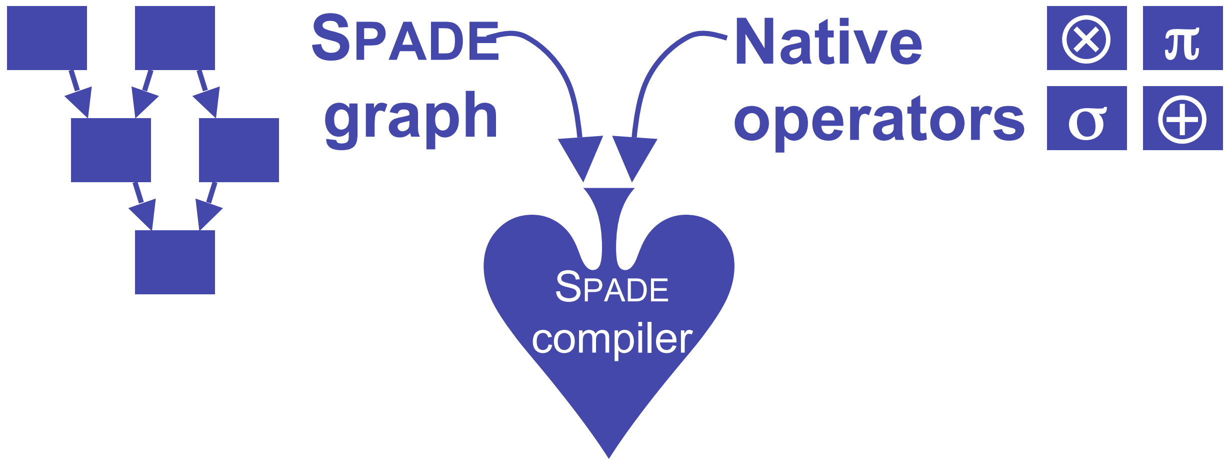
Composite operator	**Primitive operator**
Encapsulates SPADE stream graph	Encapsulates imperative code
Written in SPADE	Written in native language (e.g. C++)
Invoked/customized from SPADE	Invoked/customized from SPADE
Specialized by compiler	Specialized by compiler

Composite Operator Parameters

Original graph **Composite op. M** **Expanded graph**



PL Problem Revisited



Problem:

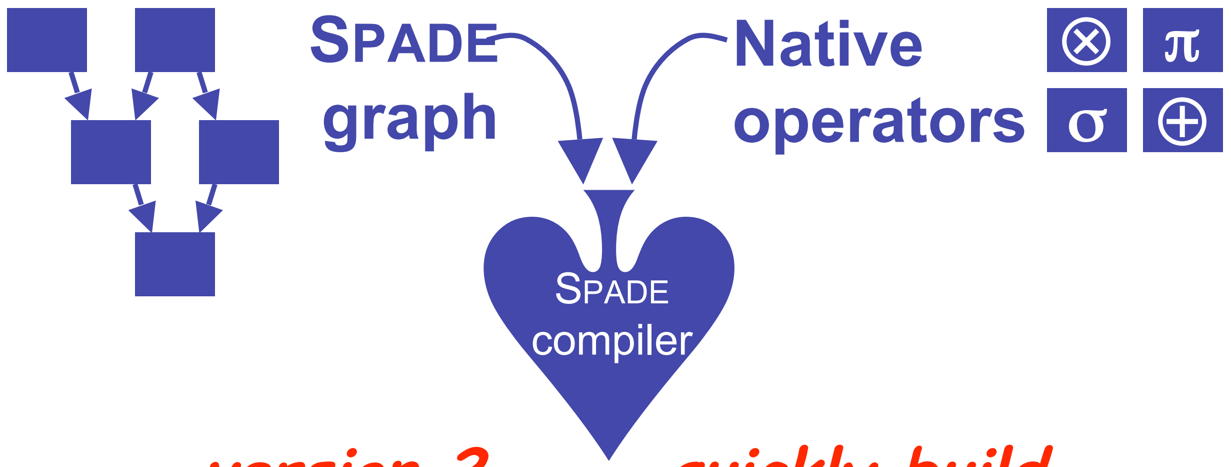
Design SPADE language and compiler.

Priorities:

Performance:	Generate specialized operator code.
Generality:	Arbitrary graphs, arbitrary C++ code.
Usability:	Composite operators, clear syntax.

PL Problem Revisited

technical and nontechnical



Problem:

version 2 quickly build

Co-Design SPADE language and compiler.

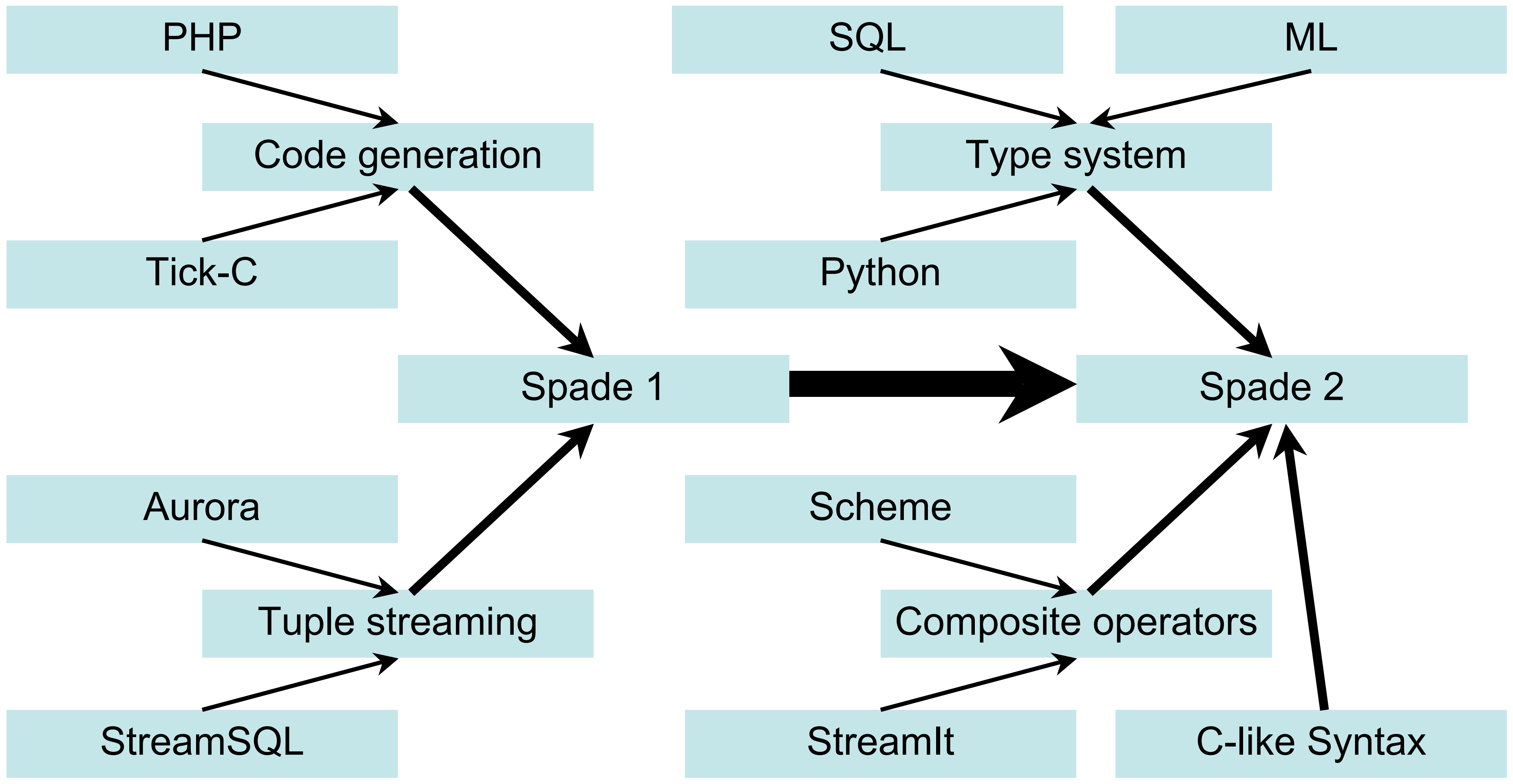
Priorities:

Performance:	Generate specialized operator code.
Generality:	Arbitrary graphs, arbitrary C++ code.
Usability:	Composite operators, clear syntax.

Outline

- Systems Solution
- Language Problem
- Language Design
- Design Process
- Future Work

Language Design: Reuse+Combine



(Incomplete map of language influences)

Language Design: Iterate

Collected and prioritized requirements	Wrote language spec, published internally	Wrote parser grammar	Feedback on spec from wiki, revised design	Feedback on spec from talk, revised some more	Designed compiler components	Published tech report, started coding
Sep ‘08	**Oct ‘08**	**Nov ‘08**	**Dec ‘08**	**Jan ‘09**	**Feb ‘09**	**Mar ‘09**

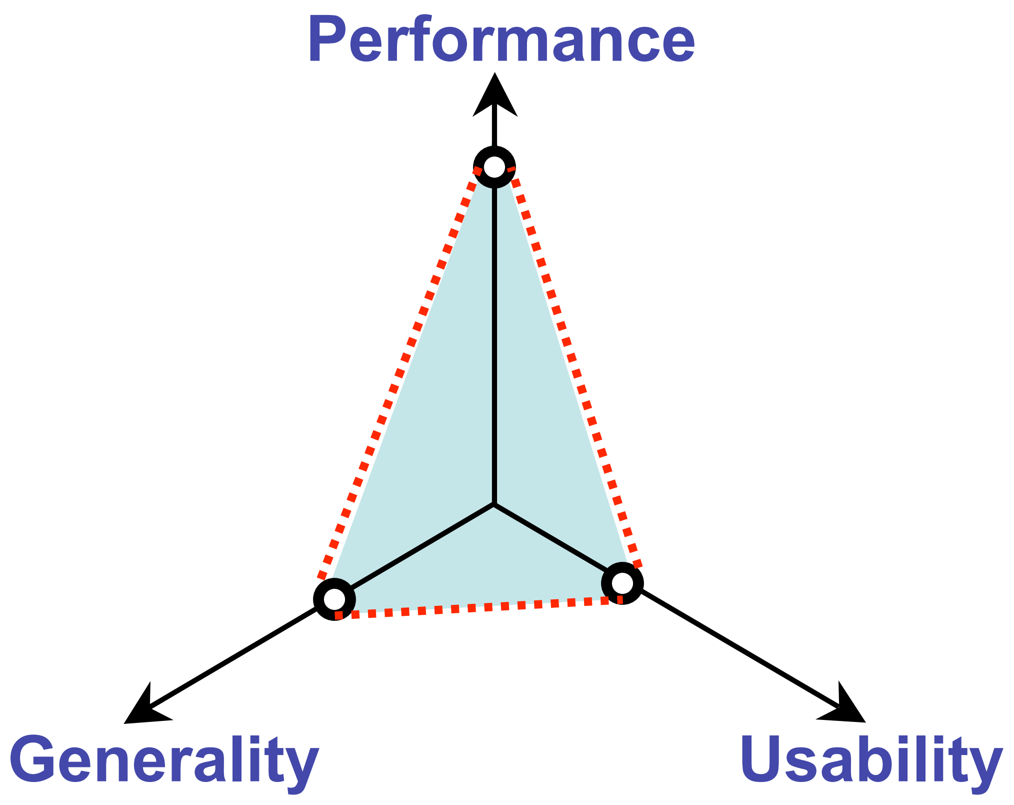
Language Design: Iterate

- Meeting preparation
  - Agenda (which features to discuss)
  - Examples (so everyone can see the issues)
- During the meeting
  - Project agenda and examples
  - Project meeting notes (decisions <u>and</u> rationale)
  - Be humble (maybe you are not right)
  - When "stuck" on an item, move on to next item
  - Wrap up meeting after 1 hour max
- After meeting, send notes to everyone

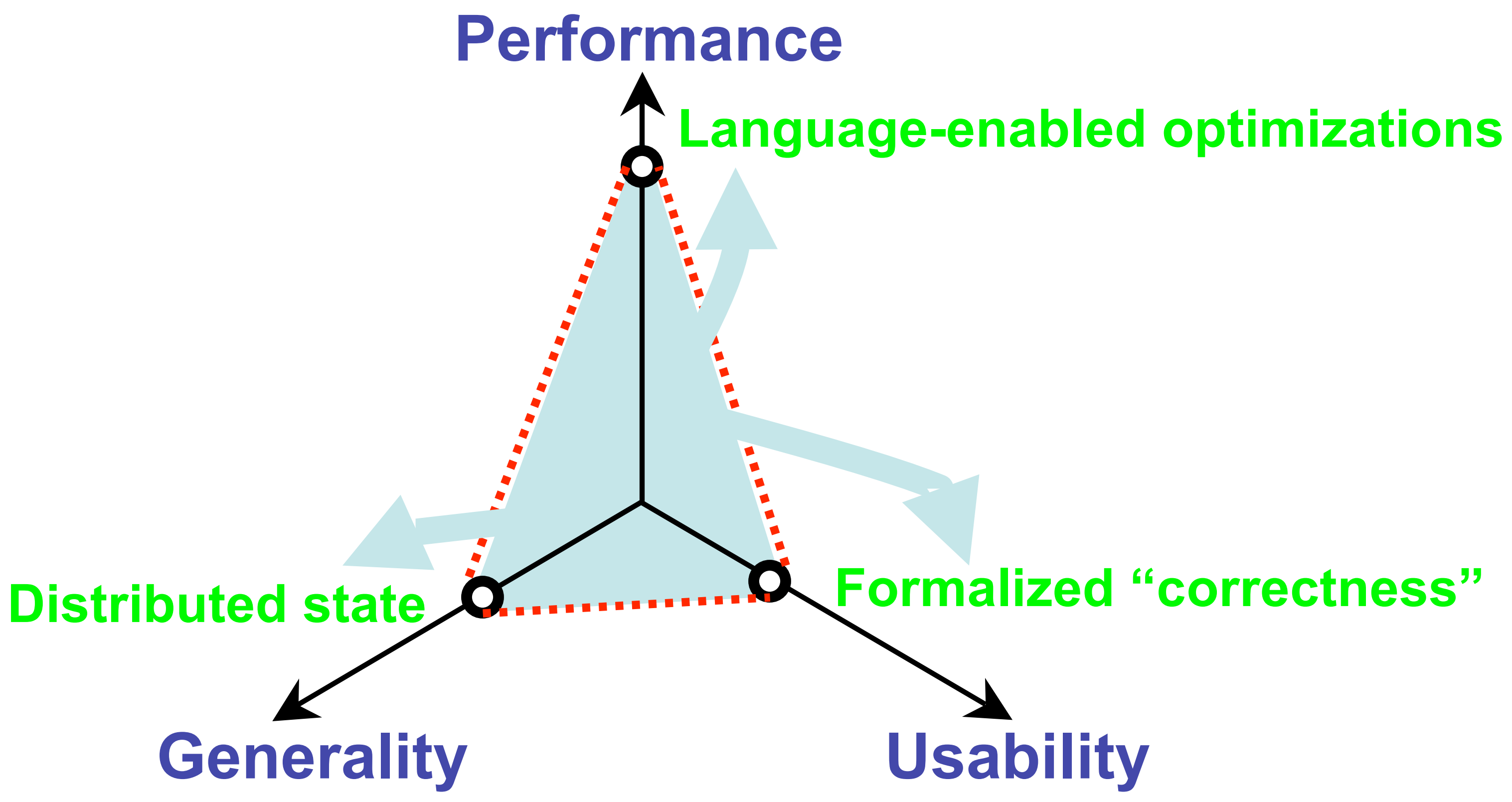
Outline

- Systems Solution
- Language Problem
- Language Design
- Design Process
- **Future Work**

Design Goals and Limits



Opening the Levees



Conclusions

- Emerging technologies are language design opportunities.
- Practical language design: reuse, combine, iterate.
- Language specification available as TR.