

# Programming Crystalline Hardware

Phillip Stanley-Marbell

Diana Marculescu

Dept. of ECE, Carnegie Mellon

{pstanley, dianam}@ece.cmu.edu

# Outline

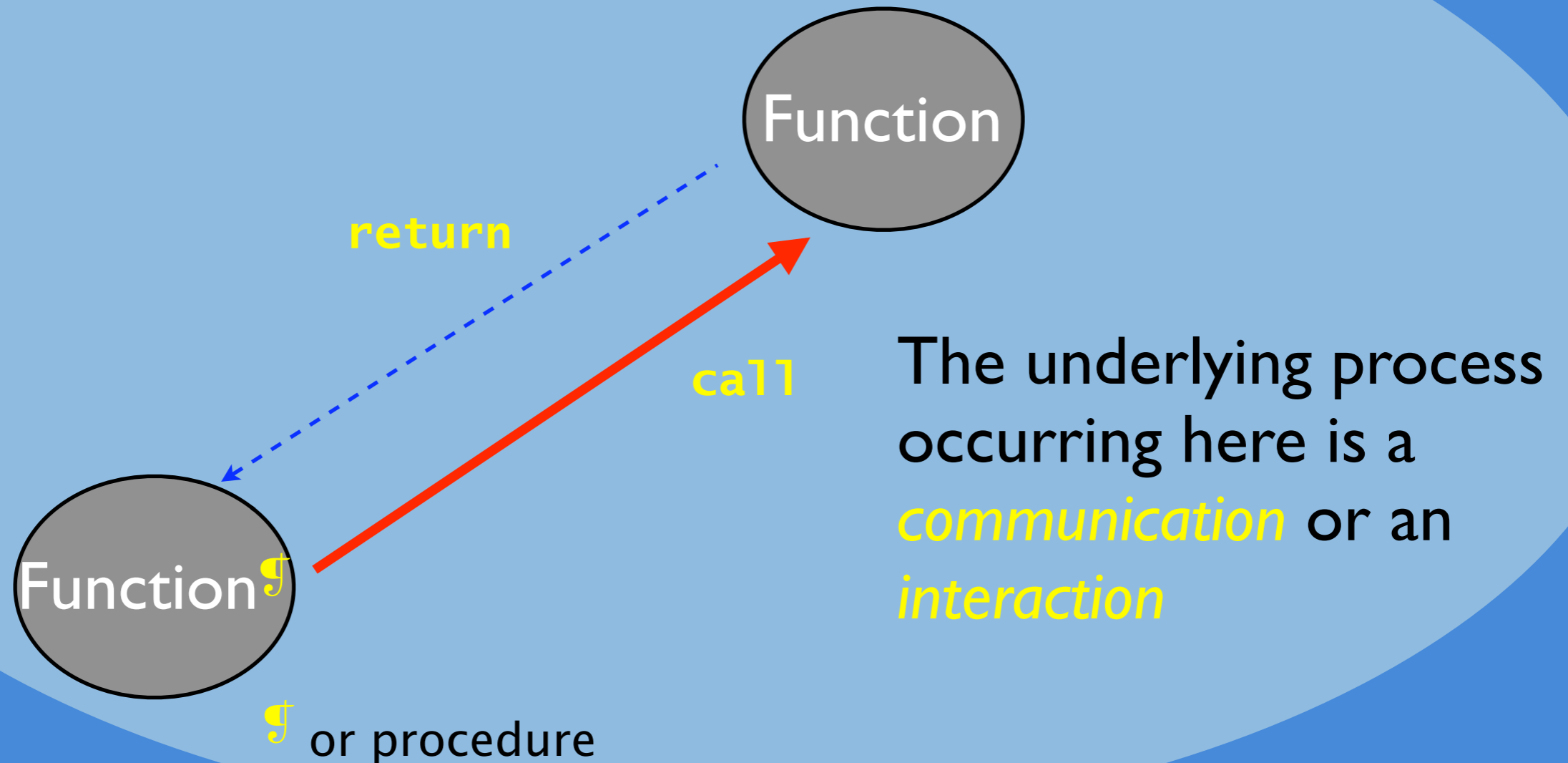
- Motivation & Context
- Our Proposal
- Examples and Possible Optimizations
- Related Research
- Summary

# Motivation

- Hardware used to be expensive,
  - Communication was (relatively) cheap; Multiplex irregular hardware in time
- Interface to time-multiplexed hardware is the *instruction*
  - Programs structured around change in control flow
- Hardware became cheaper, defect prone
  - CAEN hardware
  - Communication is increasingly expensive, possibly error-prone
  - **Employ regularity to achieve defect and runtime fault tolerance**
- 
- How to program failure-prone regular hardware ?
  - **Expose communication for reliability optimization**

# Communication vs. Control Flow

## Program



- **Errors in the interaction : errors in communication**
  - Some applications might be able to tolerate errors in this interaction

# Observation

- Applications made up of modular units (e.g., functions)
- Interact by transferring flow of control
  - A vestige of constraints originally imposed by hardware
- **Underlying phenomenon** occurring is actually an *exchange of information* or *interaction*
- **Structure programs to make communication explicit**
  - Employ information-theoretic techniques to make interactions reliable in the presence of errors

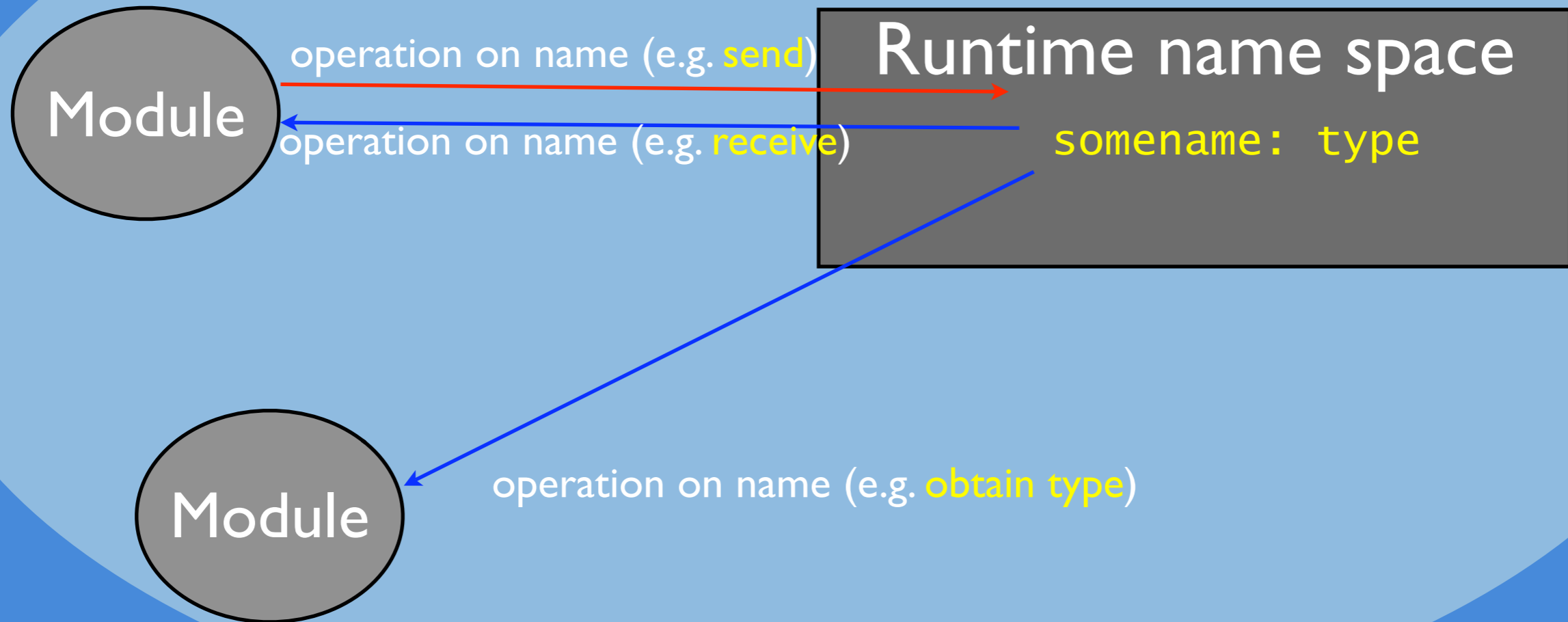
# Our Proposal

(Investigating these ideas in a prototype language, **M**)

- Programs structured as a collection of *modules*
- Interface to modules is a *name*
- Modules interact by *communication on names*
- Names have *types*
- Typed names arranged in a *name space*
- *Interaction on names defined on a small set of operators*

# Proposal

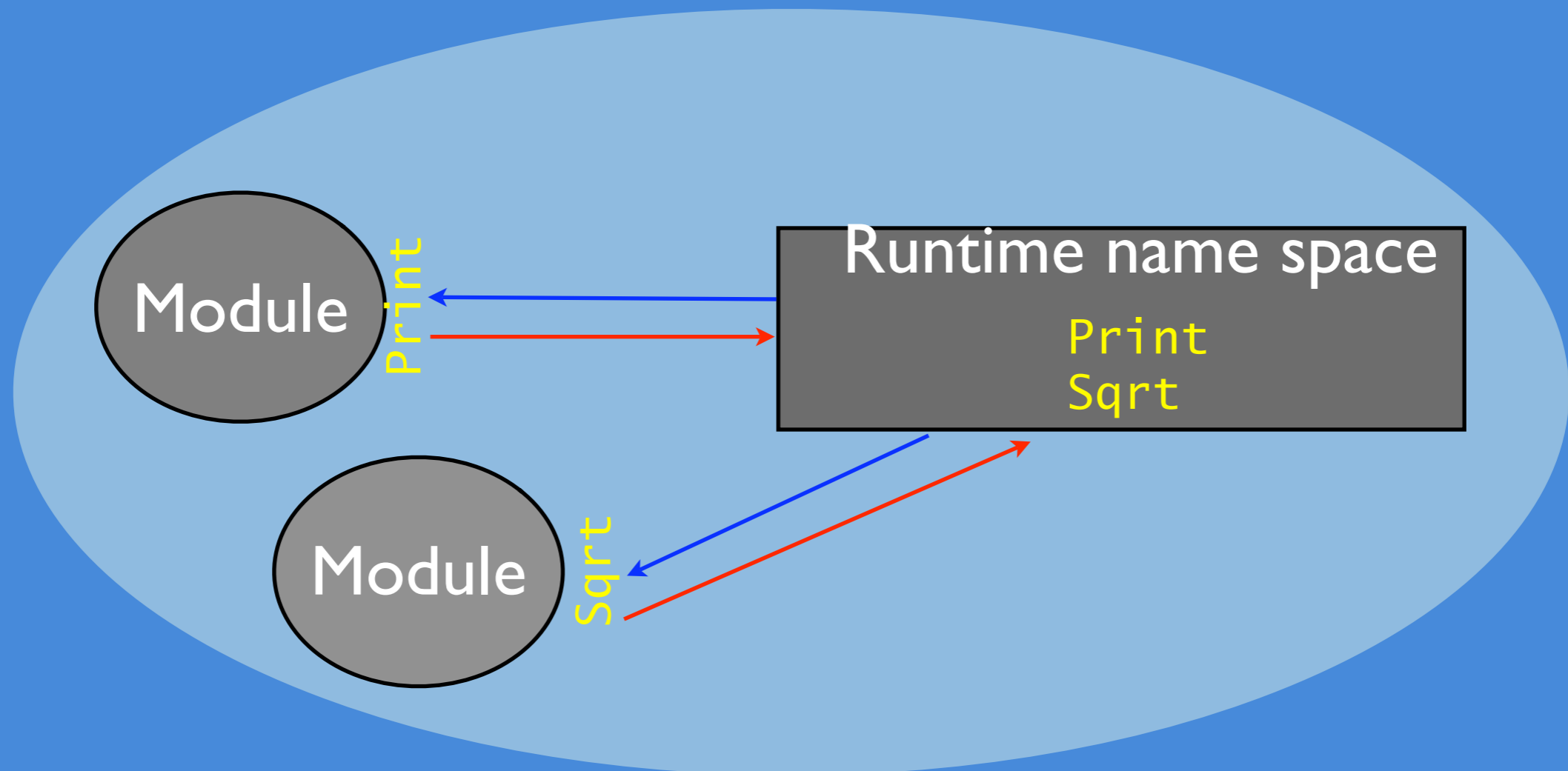
## Program



*Modules communicate through names in runtime name space*

# Modules

- Programs are collections of *modules*
  - No transfer of control flow between modules
  - Interface to a module is a *name*
  - **Example:**
    - A module, **Print**, that employs another module, **Sqrt** to compute square root



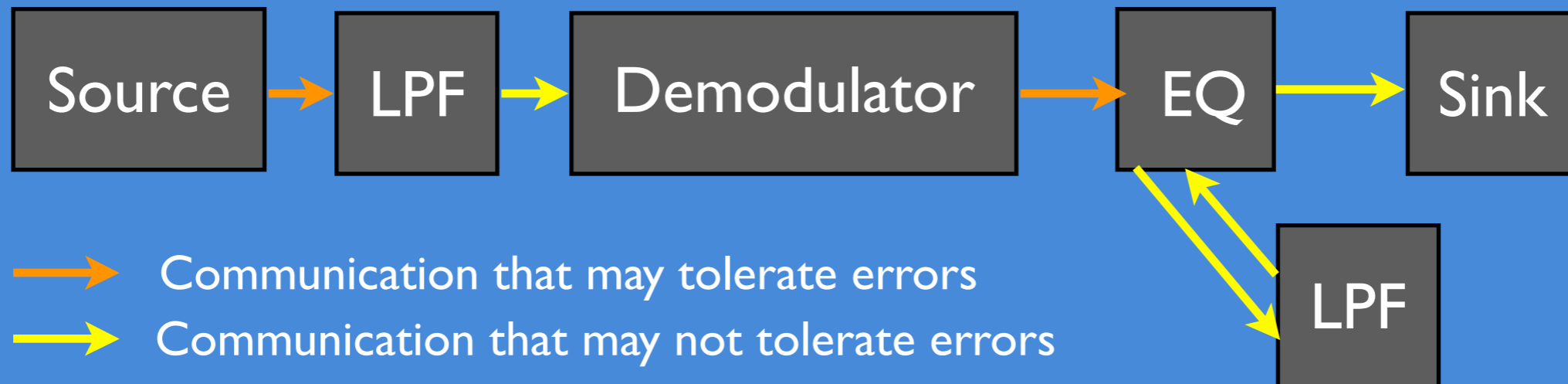
# Names & Name Operators

- Names represent modules and channels in programs
- A small set of **operators for communicating on names**
  - **nameread** — Receive from a name
  - **namewrite** — Send on a name
  - **name2chan** — Bind a name to a variable
  - **name2type** — Obtain description of name's type
  - **chan2name** — Bind a variable to a name
- Actual **encoding** of operations specified at compile time

# Encoding Name Operations

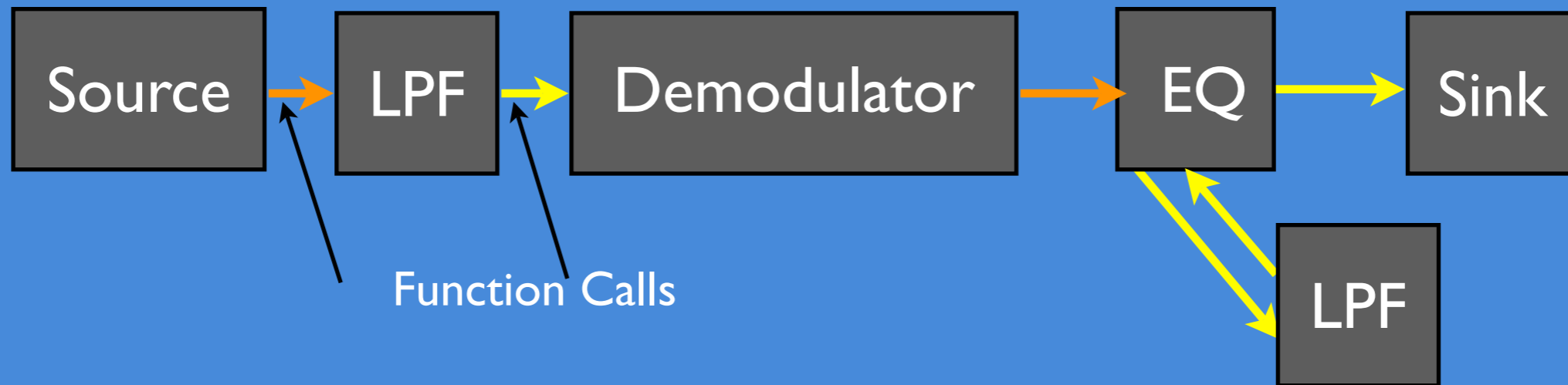
- Operations on names are pre-defined
  - `nameread`, `namewrite`, `name2chan`, `name2type`
- How these operations will be encoded (i.e., bits) is a compilation-time decision
- Both ends of communication must use same encoding
- Specifying encoding is, in essence, defining correspondence b/n a pattern of bits and one of the operation types
- Tradeoff in employing more bits for reliability vs. overhead

# Example: Software Radio



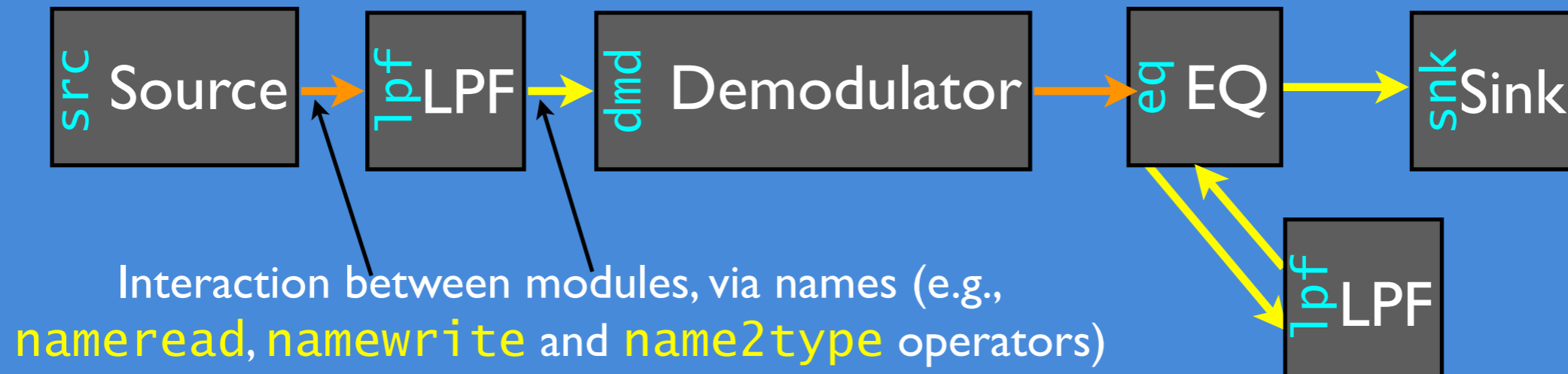
- Comprised of 5 major components
- Components exchange data (communicate)
  - Some communications can tolerate error e.g. b/n Source and LPF
  - Some communications must be error free, e.g., b/n EQ and Sink
- How to map to a error-prone regular substrate ?
  - Map each of 5 components to one or more units in hardware

# Software Radio with Functions



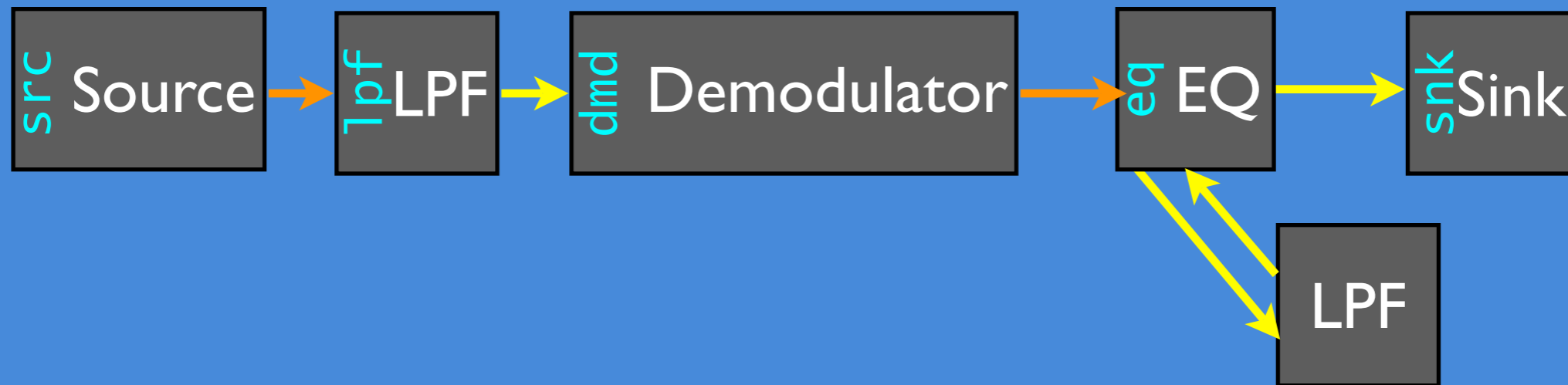
- Each stage represented by a function
- Functions interact by call/return (change in control flow)
- Data copies reduced by passing pointers b/n functions
- Difficult to reason about optimizations for reliability

# Software Radio in M



- Each module (Source, LPF, Demodulator, EQ, Sink) represented by a name in runtime name space
- Type of name represents interface of module
- Modules interact by performing operations on names
- Possible optimizations for performance and reliability

# Software Radio in M



- Optimization for reliability:
  - Employ an appropriate encoding for operations on names
  - Operations (e.g., **nameread**, **namewrite**) on the name **lpf** could be encoded with fewer bits (errors between **Source** and **LPF** modules can be tolerated)
- Reducing Data copies
  - Use Huffman codes and an appropriate codebook to in lieu of pointers ?

# Related Research

- A uniform name space for module/resource interaction
  - Linda/Tuple spaces [Carreiro & Gerlenter, '89]
  - Plan 9 [Pike et al., '95]
- Programs structured around I/O
  - CSP [Hoare, CACM '78]
  - Occam [May, '84]
- Formalism for underlying computational model
  - CCS,  $\pi$ -Calculus [Milner, '80]
  - Ambients [Cardelli & Gordon '98]
- Programming both crystalline and amorphous hardware
  - StreamIt [Gordon et al., ASPLOS '02]
  - Programming a “paintable computer” [Butera, PhD thesis, '02]
  - Programming methodology for self-assembling systems [Nagpal et al., AAAI '02]

# Summary

- **Regular hardware substrates**
  - For defect tolerance in CAEN hardware substrates
  - To mitigate increasing costs of wires in traditional VLSI designs
  - Motivates communication exposed software
  - **Error prone** computational and communication substrate
- **Underlying all module interactions is *communication***
- **Software for regular failure-prone substrate ?**
  - Programs organized as collection of **modules**, interact by explicit communication
  - Communication between modules occurs on **names**
  - Small set of **operators** on which communication is defined
  - **Optimizations on name operations for performance and reliability**
- **Typos in paper**
  - Update @ <http://www.ece.cmu.edu/~pstanley/nsc2-paper.pdf>

# Thank You

*NSC-2, San Diego CA*