Advanced Analysis and Design
Architectural Styles

Professor David S. Rosenblum
Department of Computer Science
http://www.cs.ucl.ac.uk/staff/D.Rosenblum/

Architectural Styles

‘A set of design rules that identify the kinds of components and connectors that may be used to compose a system or subsystem, together with local or global constraints on the way the composition is done’

— Shaw & Clements, 1996

- A *family or class of architectures* sharing a common pattern of structural organization

Analogy with Civil Architecture
The Classical Style

The Pantheon
Rome, Italy

Analogy with Civil Architecture
The Gothic Style

Nôtre-Dame Cathedral
Paris, France

Analogy with Civil Architecture
The Mediterranean Style

Irvine, California, USA

Common Software Architectural Styles
Shaw & Garlan, 1996 (1)

- Dataflow Systems
  - Batch sequential
  - Pipes and filters
- Call-and-Return Systems
  - Main program and subroutines
  - Object-oriented systems
  - Hierarchical layers (onion layers)
- Independent Components
  - Communicating processes (client/server and peer-to-peer)
  - Event systems
  - Implicit invocation

Common Software Architectural Styles
Shaw & Garlan, 1996 (2)

- Virtual Machines
  - Interpreters
  - Rule-based systems
- Data-Centered Systems (Repositories)
  - Databases
  - Hypertext systems
  - Blackboards

Characterising Architectural Styles

- Component and connector characteristics
- Allowed configurations
- Underlying computational model
- Stylistic invariants
- Common examples of its use
- Advantages and disadvantages
- Common specialisations

The Pipe-and-Filter Style

The primary architectural style supported by UNIX

- Components
  - Individual programs transforming input data to output data
- Connectors
  - Unidirectional or bidirectional data streams
- Configurations
  - Parallel linear composition of program invocations
- Underlying computational model
  - Sequential data flow and transformation
- Stylistic invariants
  - Every component has one input predecessor connector and one output successor connector
- Common specializations
  - **Pipelines:** *single* linear composition of pipes and filters
  - Bounded pipes, typed pipes

Pipe-and-Filter Example
UNIX Text Processing


```
% pic mydoc.t | eqn | tbl | troff | lpr
```

Legend: *Component* *Connector*

Pipe-and-Filter
Advantages and Disadvantages

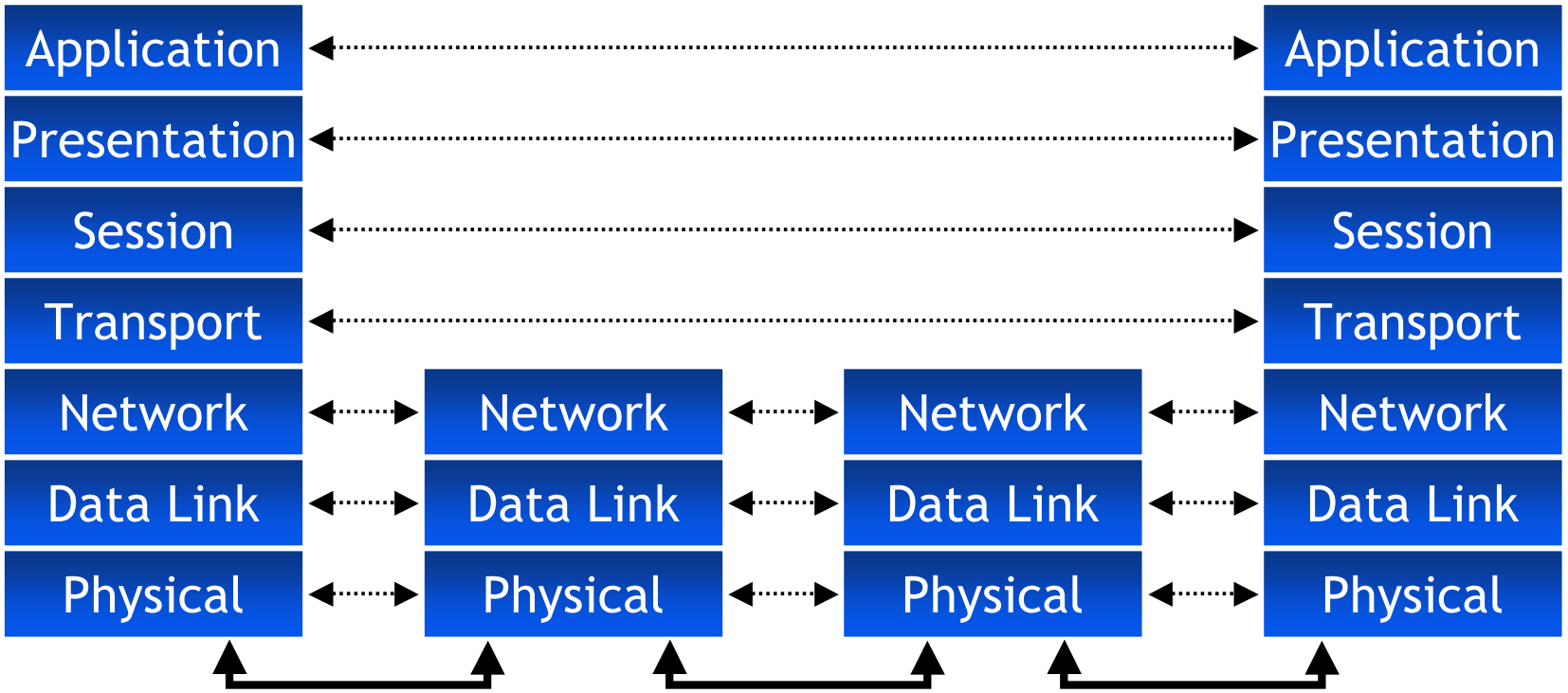
- Advantages
  - Simple, intuitive, efficient composition of components
  - High potential for reuse
  - Easy to evolve and enhance
  - Potential for limited amount of concurrency

- Disadvantages
  - Batch-oriented processing
  - Must agree on lowest-common-denominator data format
  - Limited application domain: stateless data transformation

The Layered System Style

- Components
  - Programs or subprograms
- Connectors
  - Procedure calls or system calls
- Configurations
  - ‘Onion’ or ‘stovepipe’ structure, possibly replicated
- Underlying computational model
  - Procedure call/return
- Stylistic invariants
  - Each layer provides a service only to the immediate layer ‘above’ (at the next higher level of abstraction) and uses the service only of the immediate layer “below” (at the next lower level of abstraction)

Layered System Example
OSI Protocol Stack



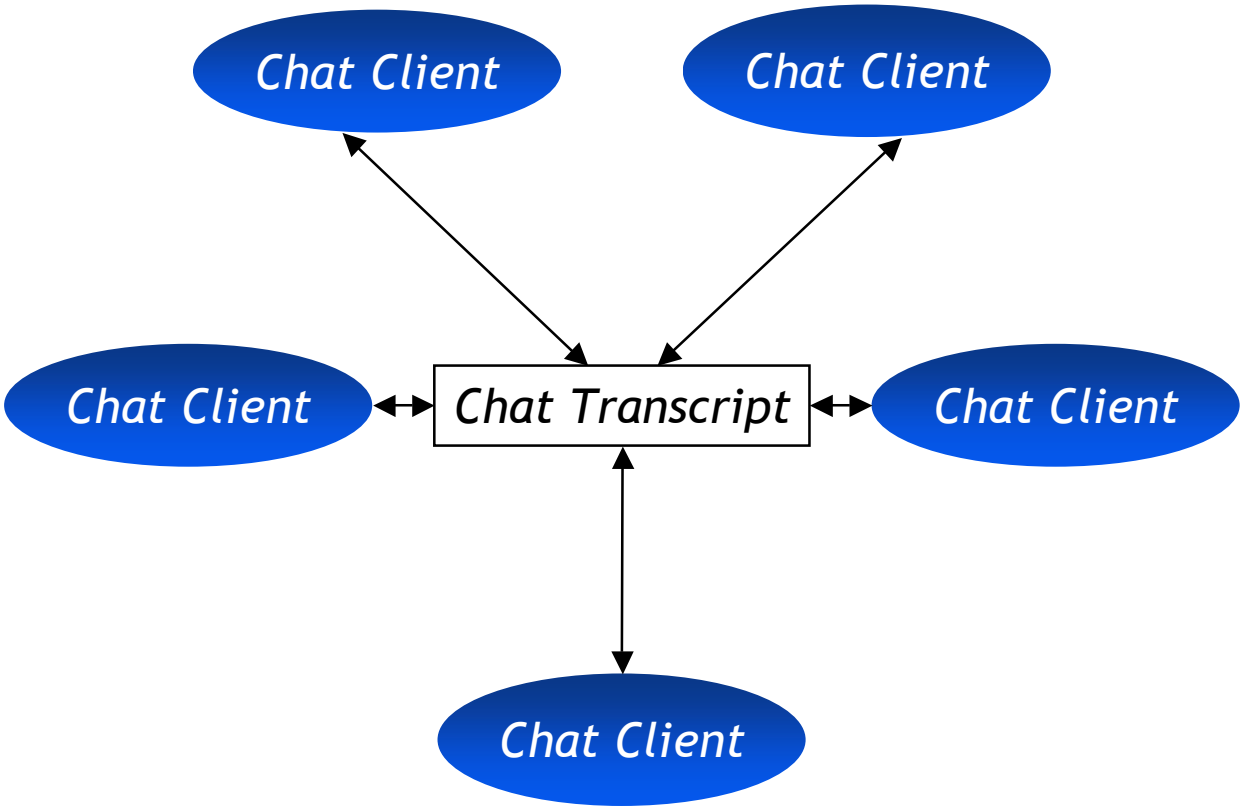
Layered System
Advantages and Disadvantages

- Advantages
  - Effective separation of concerns
  - Well-defined levels of abstraction
  - Reduced impact of change when changes don't affect layer interfaces
- Disadvantages
  - Performance degrades with too many layers
  - Can be difficult to assign functionality cleanly to the 'right' layer

The Blackboard Style

- Components
  - Blackboard client programs
- Connector
  - Blackboard: shared data repository, possibly with finite capacity
- Configurations
  - Multiple clients sharing single blackboard
- Underlying computational model
  - Synchronised, shared data transactions, with control driven entirely by blackboard state
- Stylistic invariants
  - All clients see all transactions in the same order

Blackboard Example
An Online Chat Room



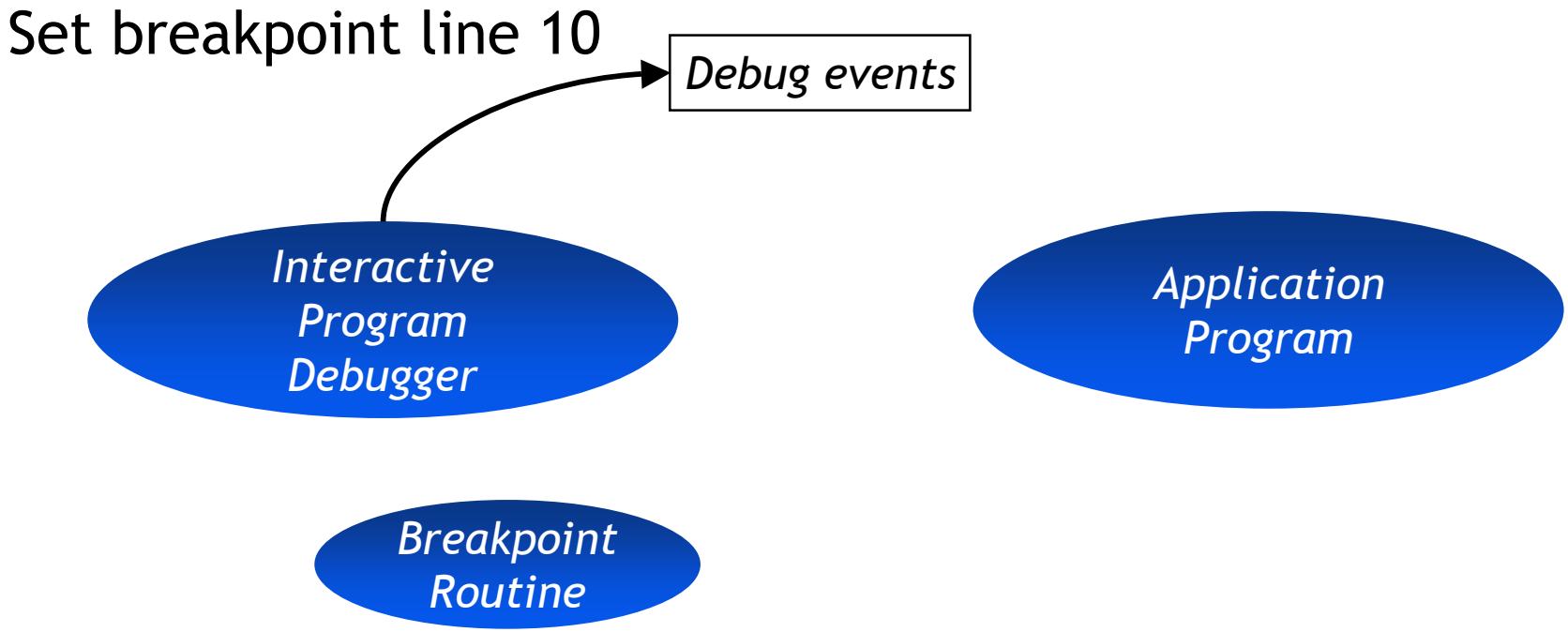
Blackboard
Advantages and Disadvantages

- Advantages
  - General style suitable for network-based applications, including network database servers
- Disadvantages
  - Blackboard becomes a bottleneck with too many clients

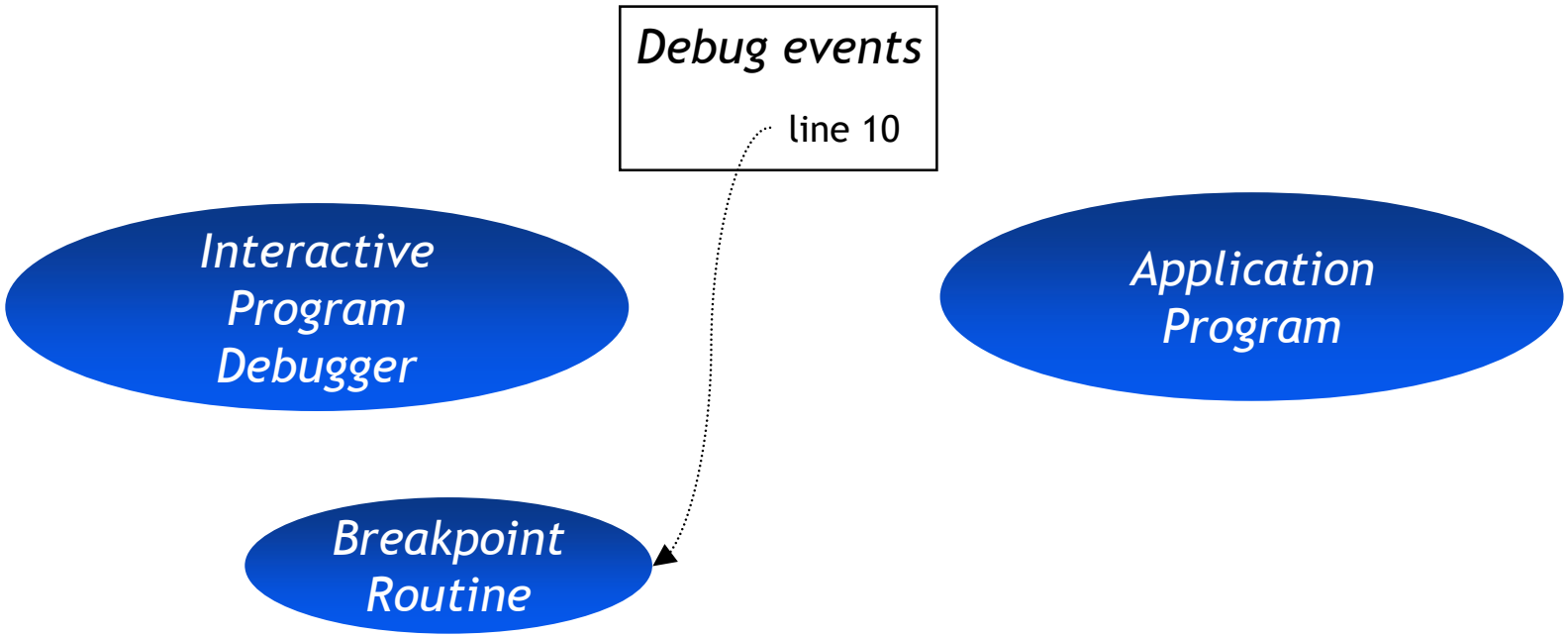
Event-Based Systems and the Implicit Invocation Style

- Components
  - Programs or program entities that *announce* and/or *register interest in* events
    - Events represent happenstances inside an entity that may (or may not) be of interest to other entities
- Connectors
  - Direct registration with announcing entities
  - *Or*, explicit event broadcast and registration infrastructure
- Configurations
  - Implicit dependencies arising from event announcements and registrations
- Underlying computational model
  1. Event announcement is broadcast
  2. Procedures associated with registrations (if any) are invoked

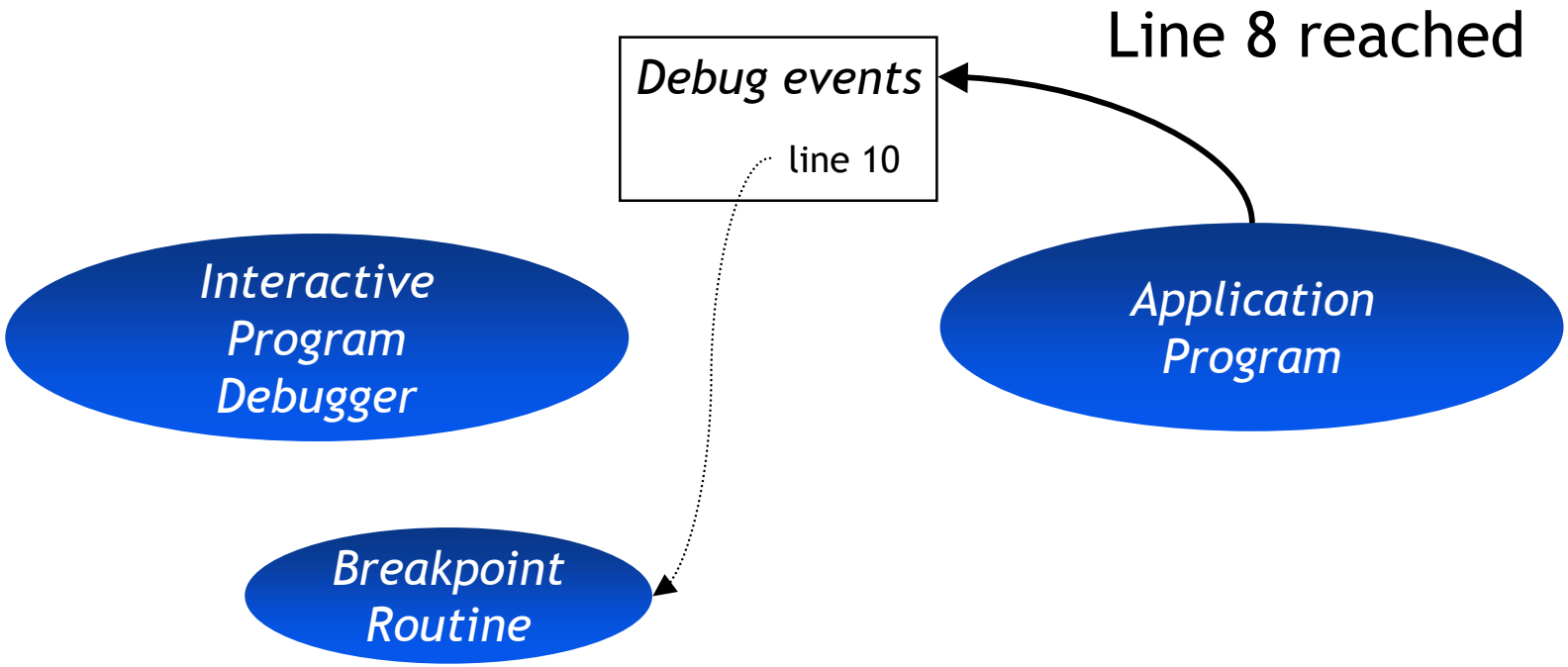
Implicit Invocation Example
Program Debugging (1)



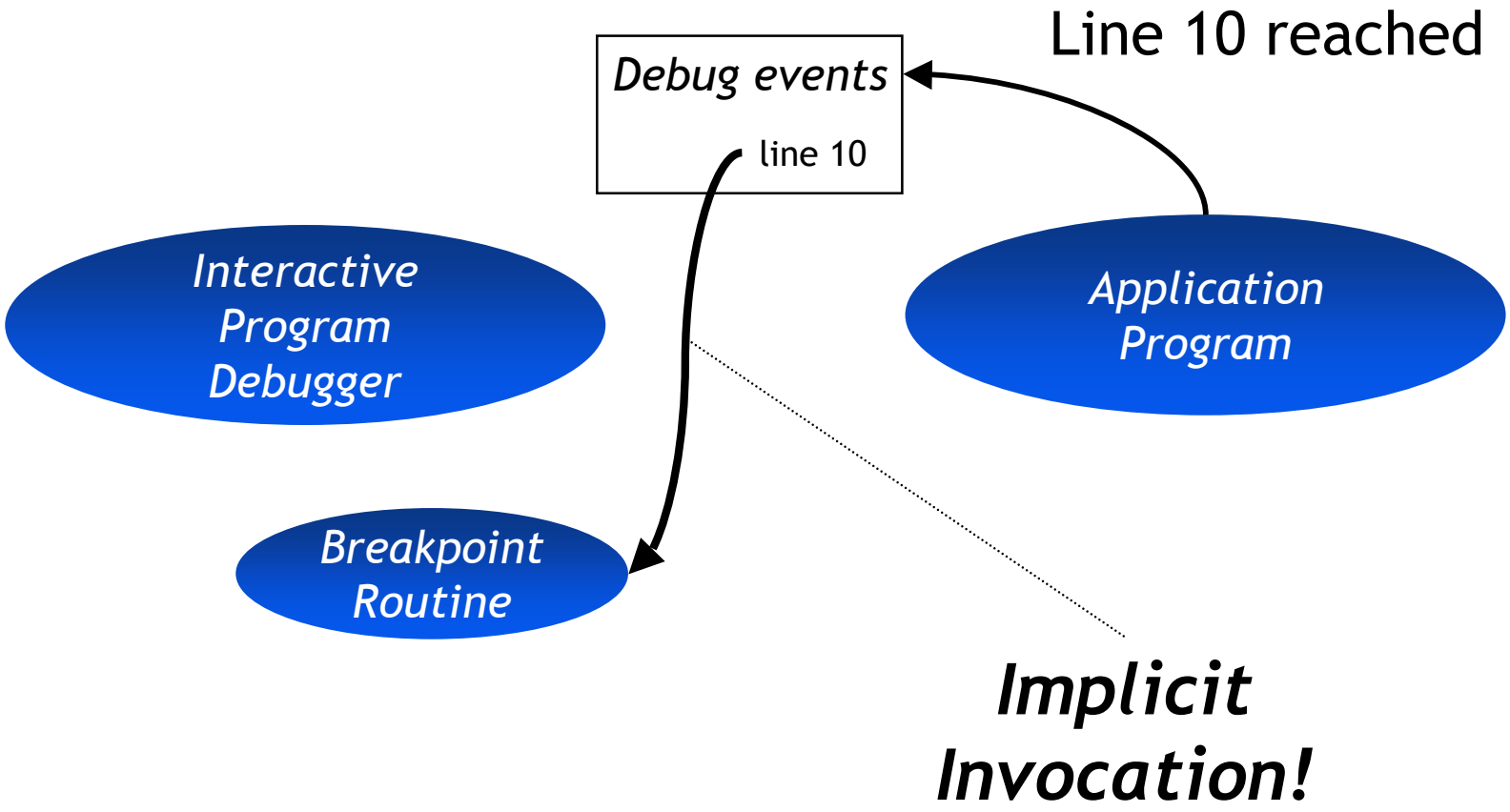
Implicit Invocation Example
Program Debugging (2)



Implicit Invocation Example
Program Debugging (3)



Implicit Invocation Example
Program Debugging (4)



Implicit Invocation
Advantages & Disadvantages

- Advantages
  - Allows for decoupling and autonomy of components
  - Enhances reuse and evolution
    - Easy to introduce new components without affecting existing ones

- Disadvantages
  - Components announcing events have no guarantee of getting a response
  - Components announcing events have no control over the order of responses
  - Event abstraction does not cleanly lend itself to data exchange
  - Difficult to reason about behaviour of an announcing component independently of components that register for its events

Some Criteria for Selecting and Comparing Styles

- Control flow
- Data flow
- Application
- Distribution
- Scalability
- *what else?*

What other styles can you think of?