Developing Embedded Software Product Lines with AspectC++

AOSD 2005 Demonstration

University of Erlangen-Nuremberg
Computer Science 4

Presenters

- **Olaf Spinczyk**
  os@aspectc.org

  University of Erlangen-Nuremberg, Germany

- **Daniel Lohmann**
  dl@aspectc.org

  University of Erlangen-Nuremberg, Germany

In this demo we present...

- AOP in deeply embedded devices
  - AOP is suitable for resource-thrifty domains!
- AOP in software product line development
  - aspects provide great benefit here!
- AspectC++ features
  - practical solutions for practical problems
- a complete Eclipse-based tool chain
  - AspectC++ Eclipse Plugin (ACDT)
  - pure::variants Eclipse Plugin

Demo Scenario

- **Embedded weather station product line**
  - sensors: wind, temperature, air pressure, ...
  - actors: display, alarm, PC connection, ...

- **Based on a small AVR ATmega μ-controller**
  - 8 Bit 4MHz RISC CPU
  - 2 – 128 kb Flash
  - 0.5 – 4 kb RAM
  - digital, analog, serial and I²C based I/O

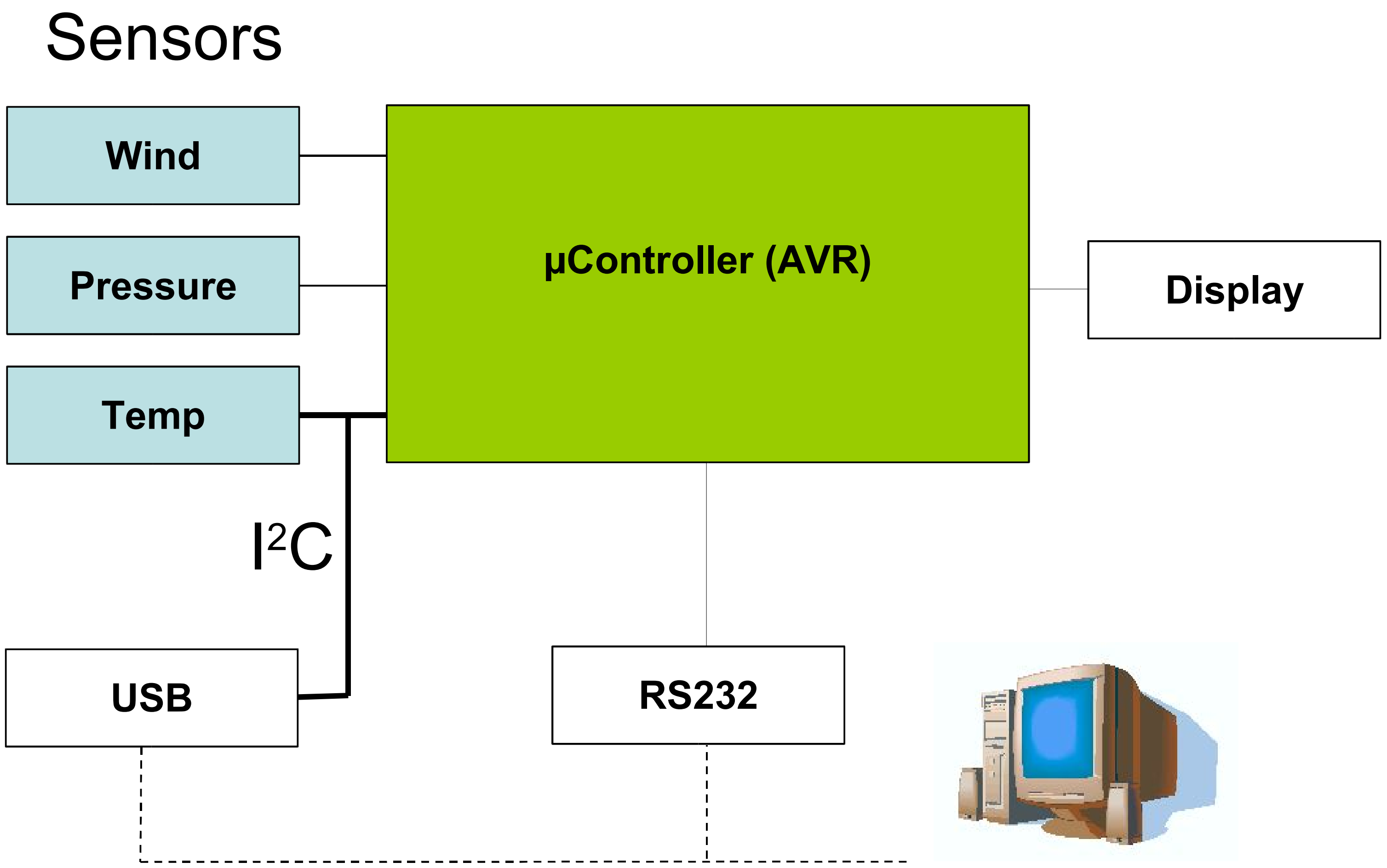
Demo Scenario

- Embedded weather station product line
  - sensors: wind, temperature, air pressure, ...
  - actors: display, alarm, PC connection, ...
- Based on a sma
  - 8 Bit 4MHz RISC
  - 2 – 128 kb Flash
  - 0.5 – 4 kb RAM
  - digital, analog, serial and I²C based I/O

AOP on this platform?

“Hello World” in AspectJ takes around **20 MB RAM** (on a PC)...

Demonstration Platform

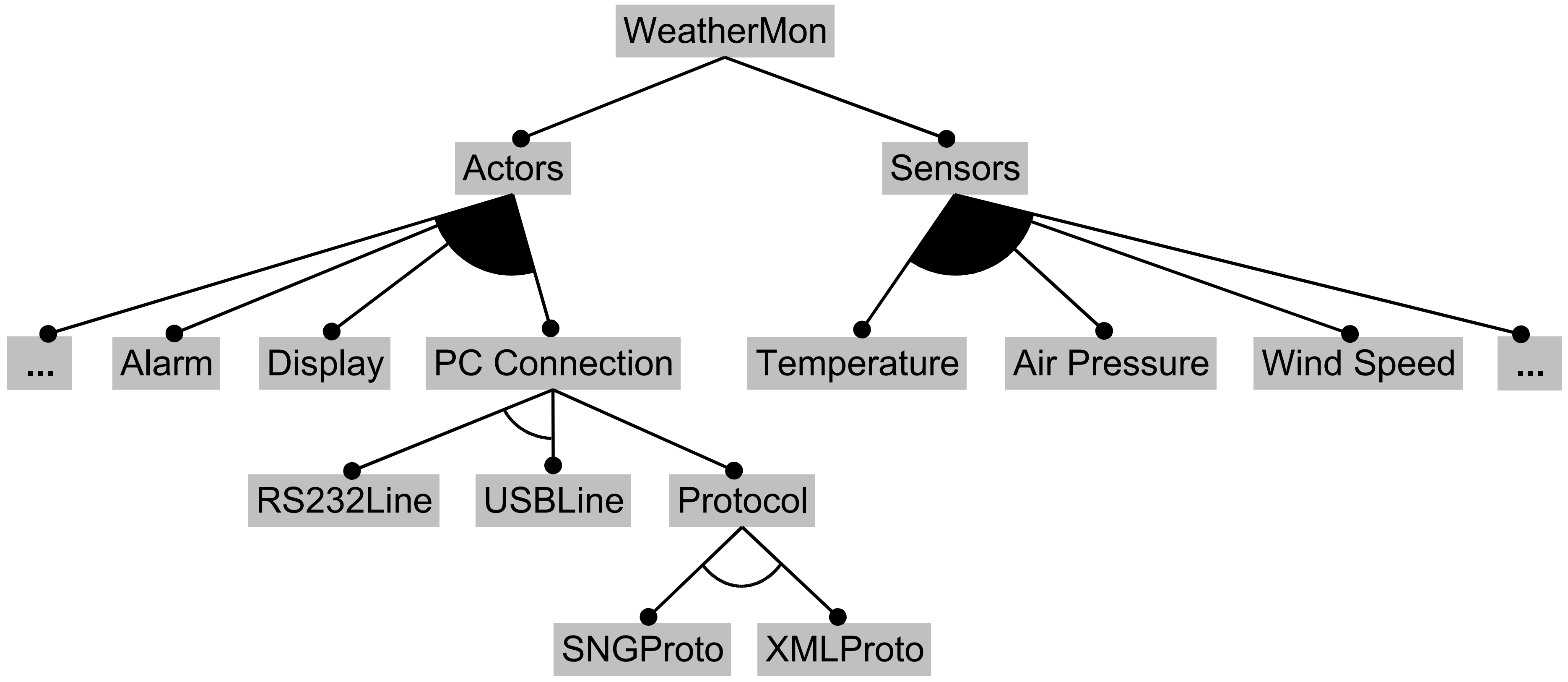


Weather Station Variants

Thermometer:	LCD, Temperature
Home:	LCD, Temperature, Pressure
Outdoor:	LCD, Temp., Pressure, Wind
Deluxe variants:	+ PC Connection
PC-only variants:	+ PC Connection - LCD

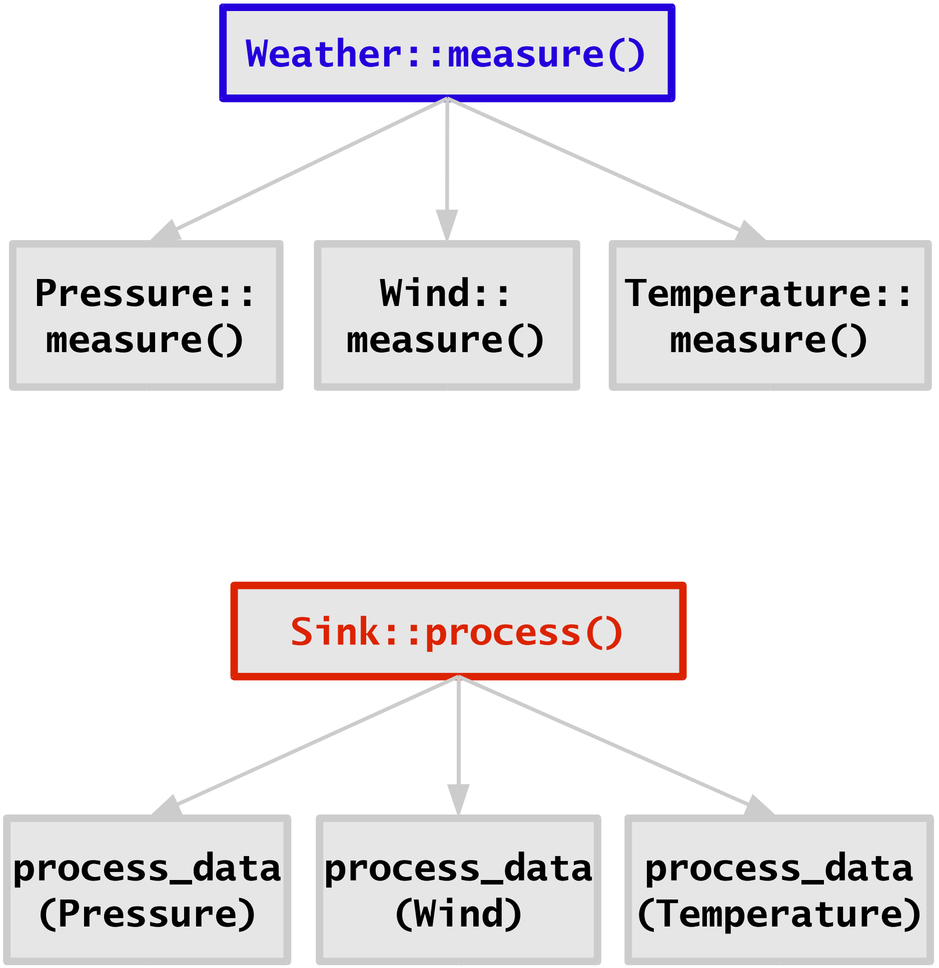
- Serial PC Connection
- USB PC Connection
- ...

Weather Station Feature Model



Weather Station: Functional Decomposition

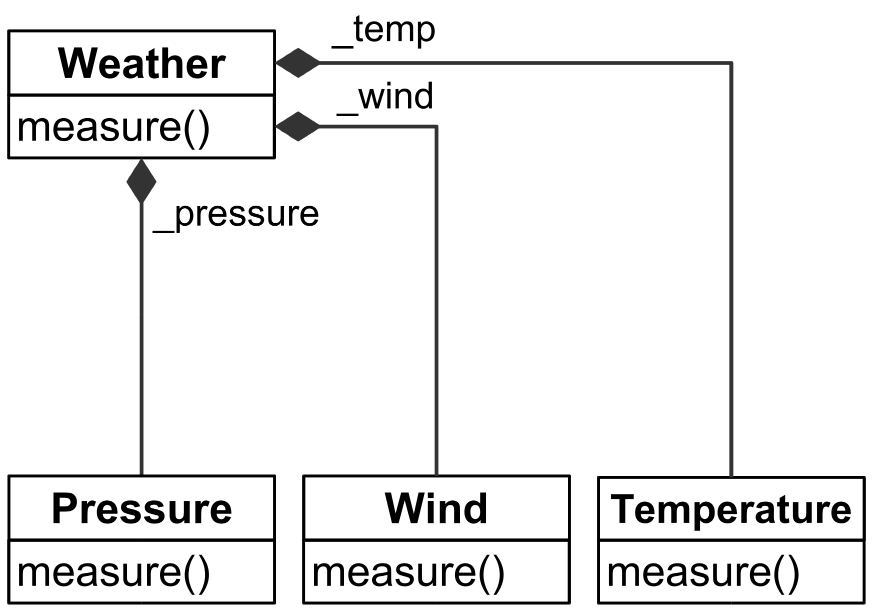
```
int main() {
  Weather data;
  Sink    sink;

  while(true) {

    // aquire data
    data.measure();

    // process data
    sink.process( data );

    wait();
  }
}
```

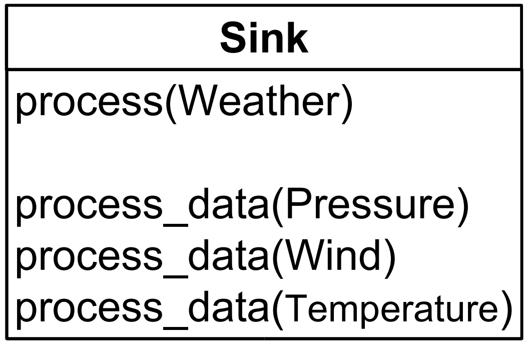


Sensor Integration

```
Weather::measure() {
  _pressure.measure();
  _wind.measure();
  _temp.measure();
}
```

```
Sink::process(const Weather& w) {
  process_data(w._pressure);
  process_data(w._wind);
  process_data(w._temp);
}
```

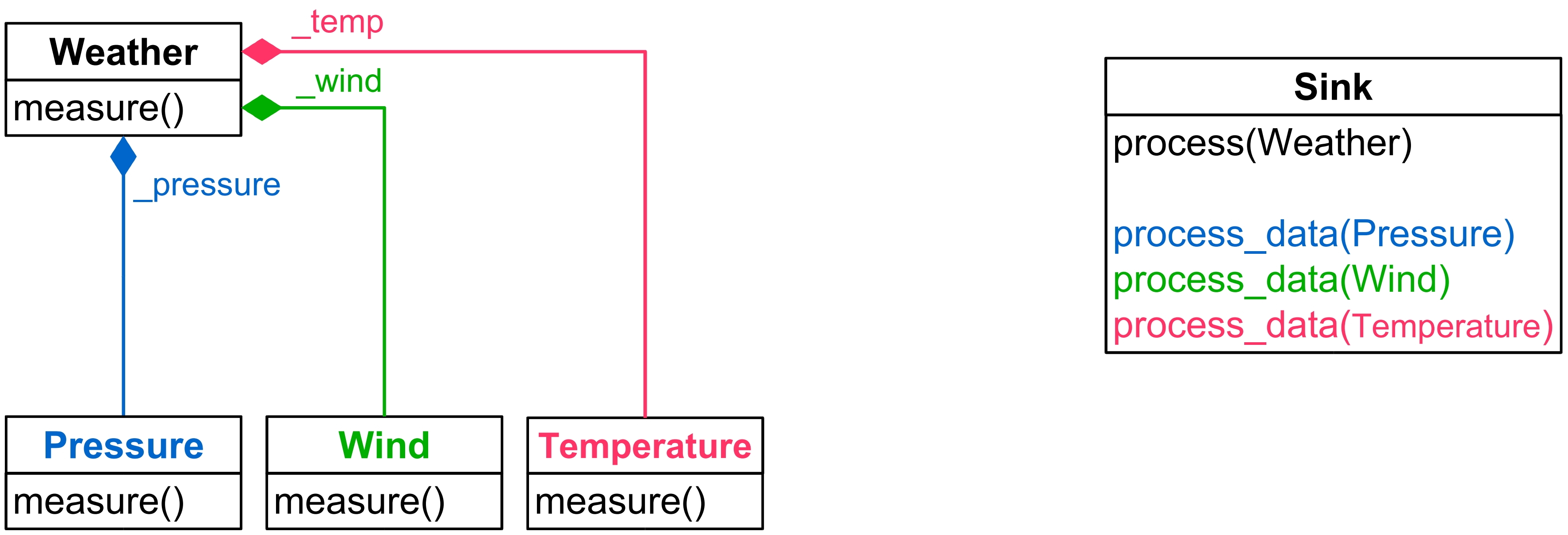




Sensor Integration

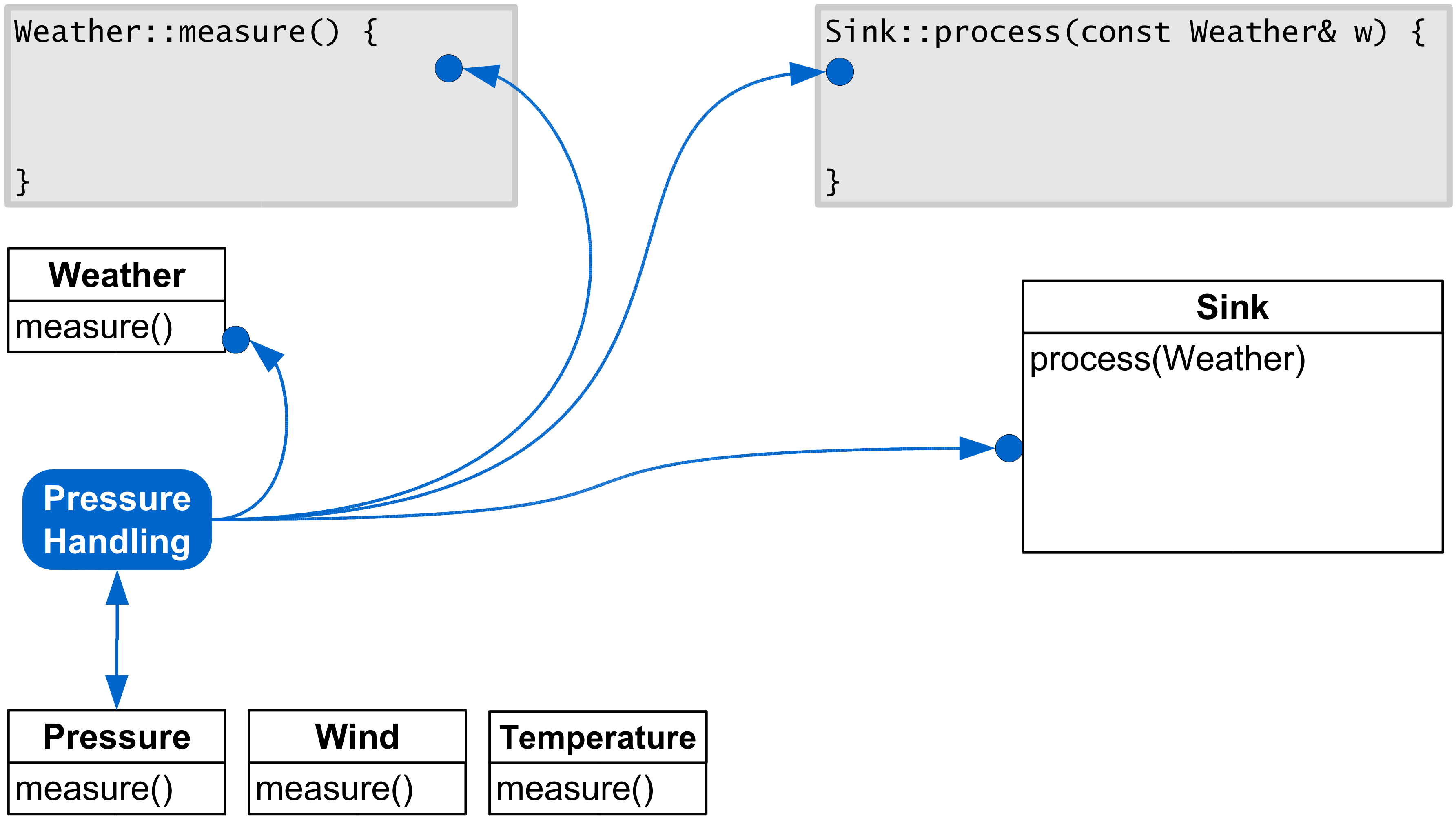
```
Weather::measure() {
  _pressure.measure();
  _wind.measure();
  _temp.measure();
}
```

```
Sink::process(const Weather& w) {
  process_data(w._pressure);
  process_data(w._wind);
  process_data(w._temp);
}
```



...crosscuts the modules

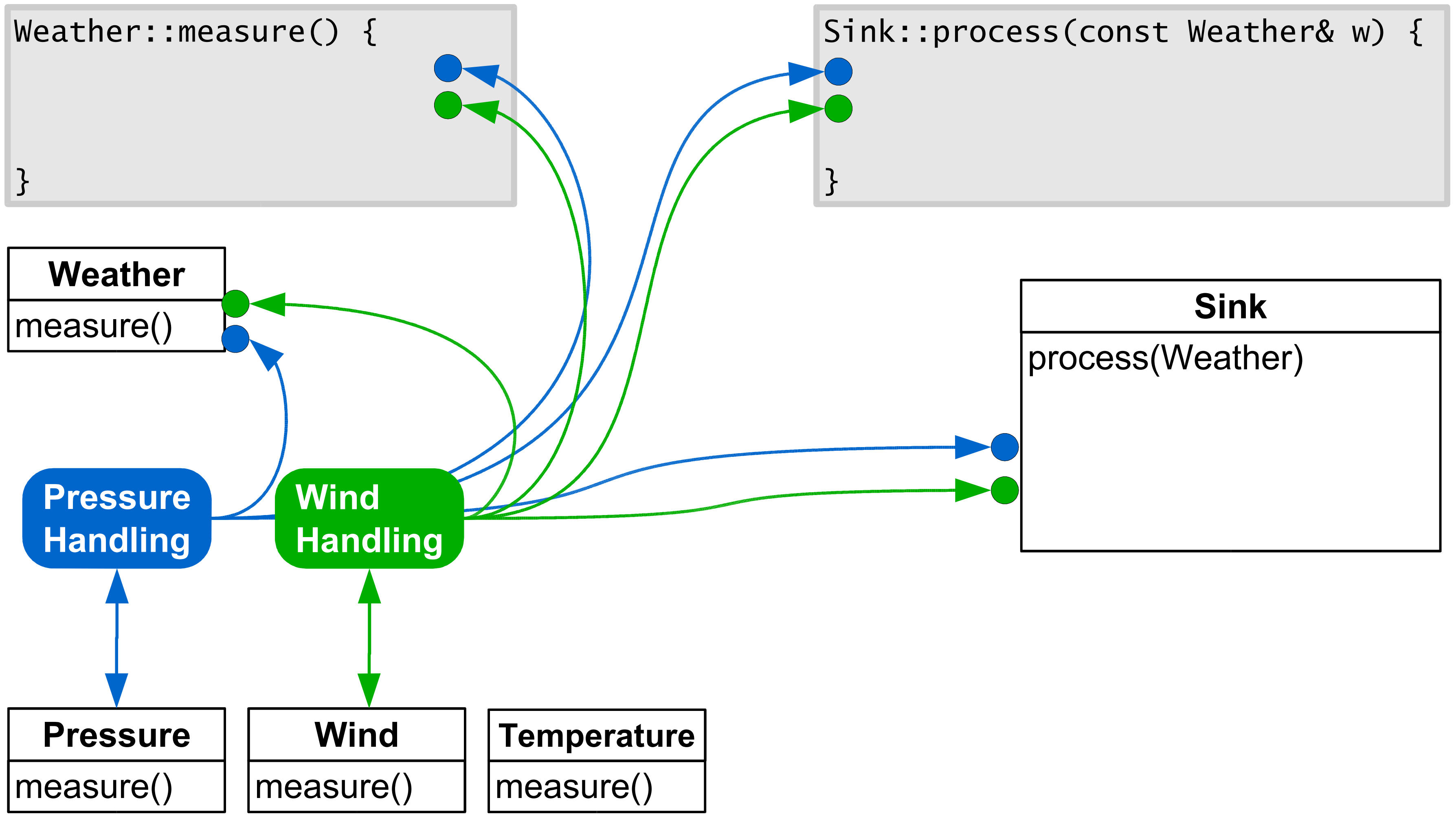
Sensor Integration with Aspects



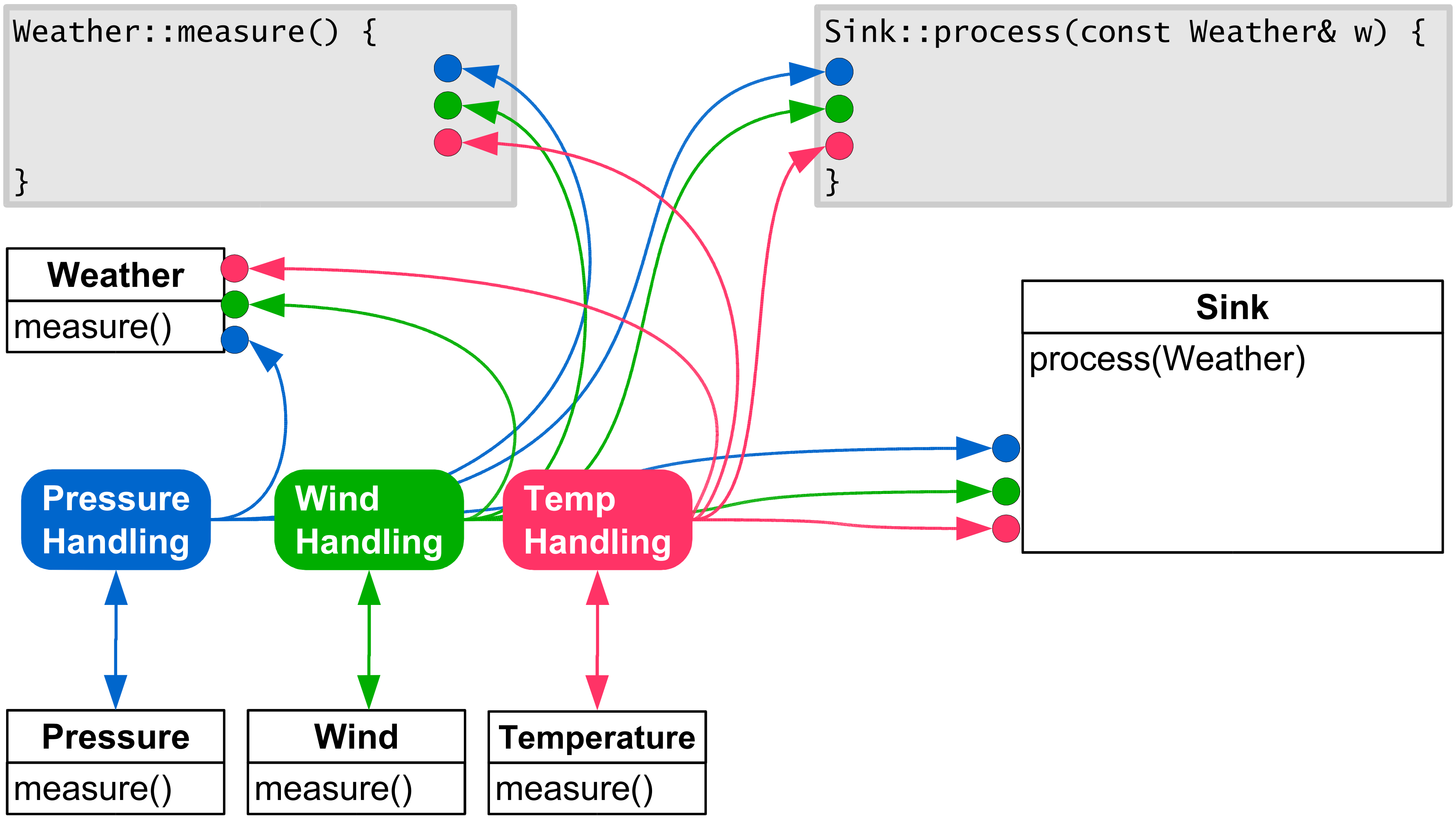
```
Weather::measure() {

}
```

```
Sink::process(const Weather& w) {

}
```

Sensor Integration with Aspects



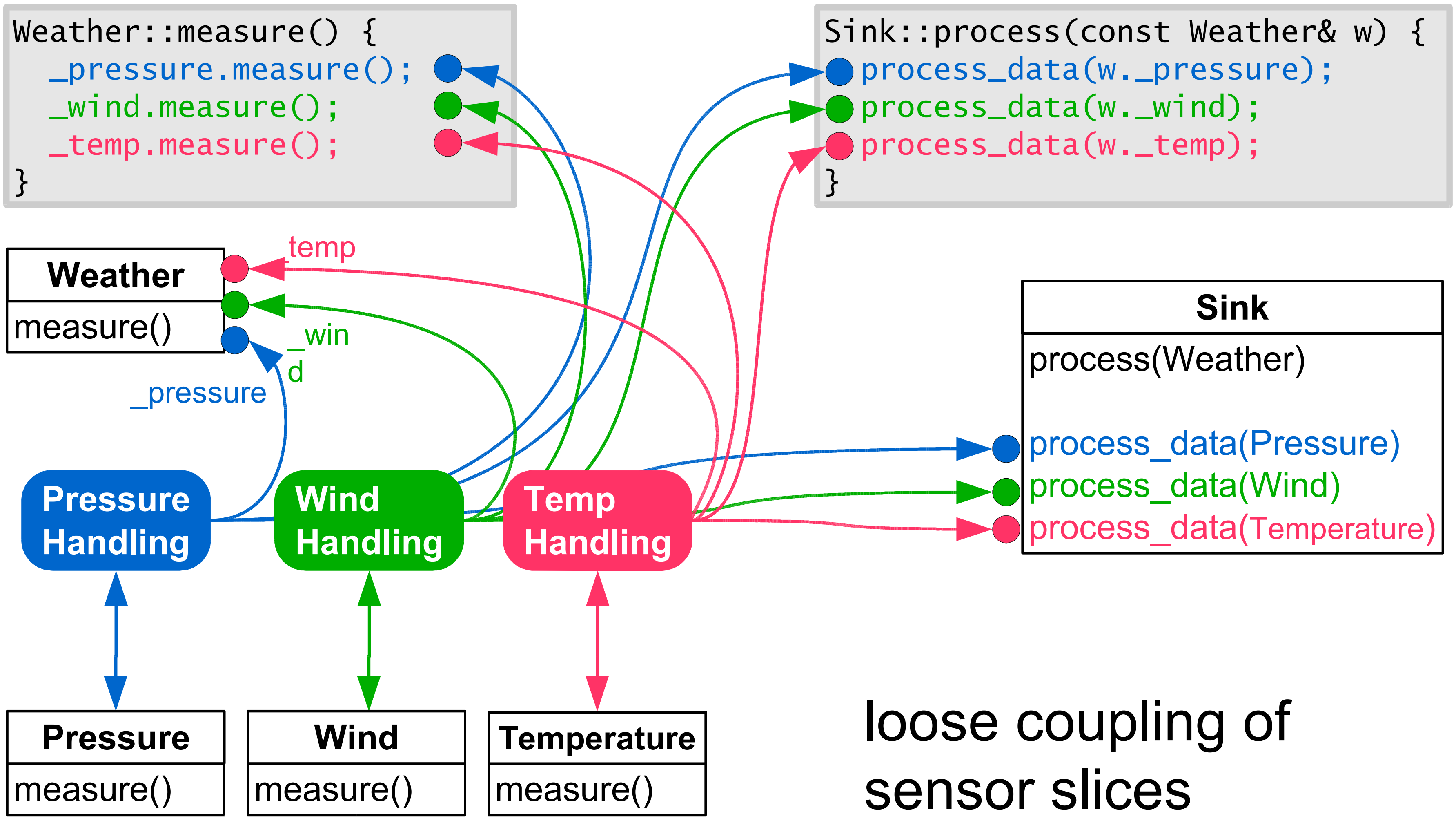
Sensor Integration with Aspects



```
Weather::measure() {

}
```

```
Sink::process(const Weather& w) {

}
```

Sensor Integration with Aspects



```
Weather::measure() {
  _pressure.measure();
  _wind.measure();
  _temp.measure();
}
```

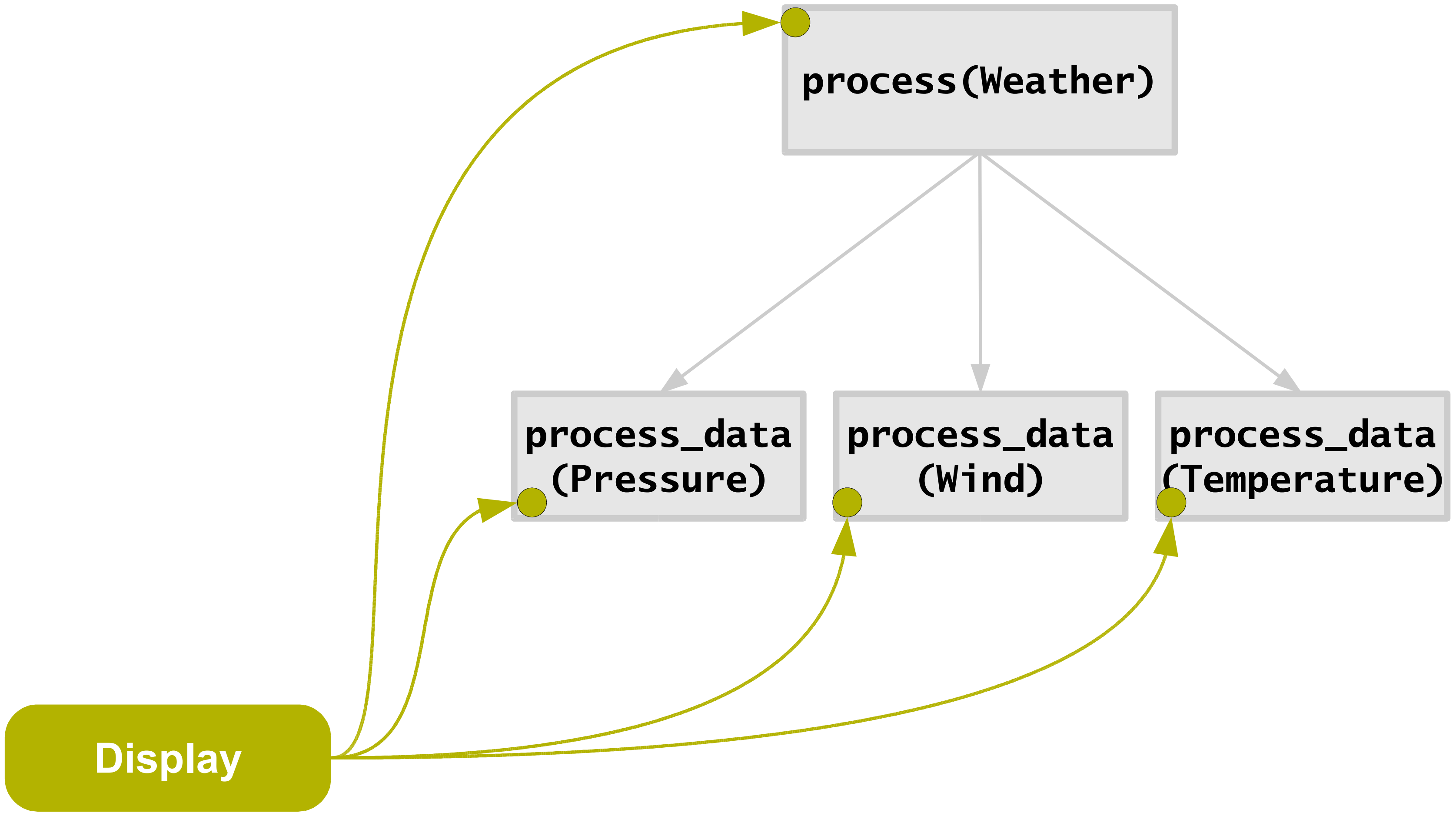
```
Sink::process(const Weather& w) {
  process_data(w._pressure);
  process_data(w._wind);
  process_data(w._temp);
}
```

loose coupling of sensor slices

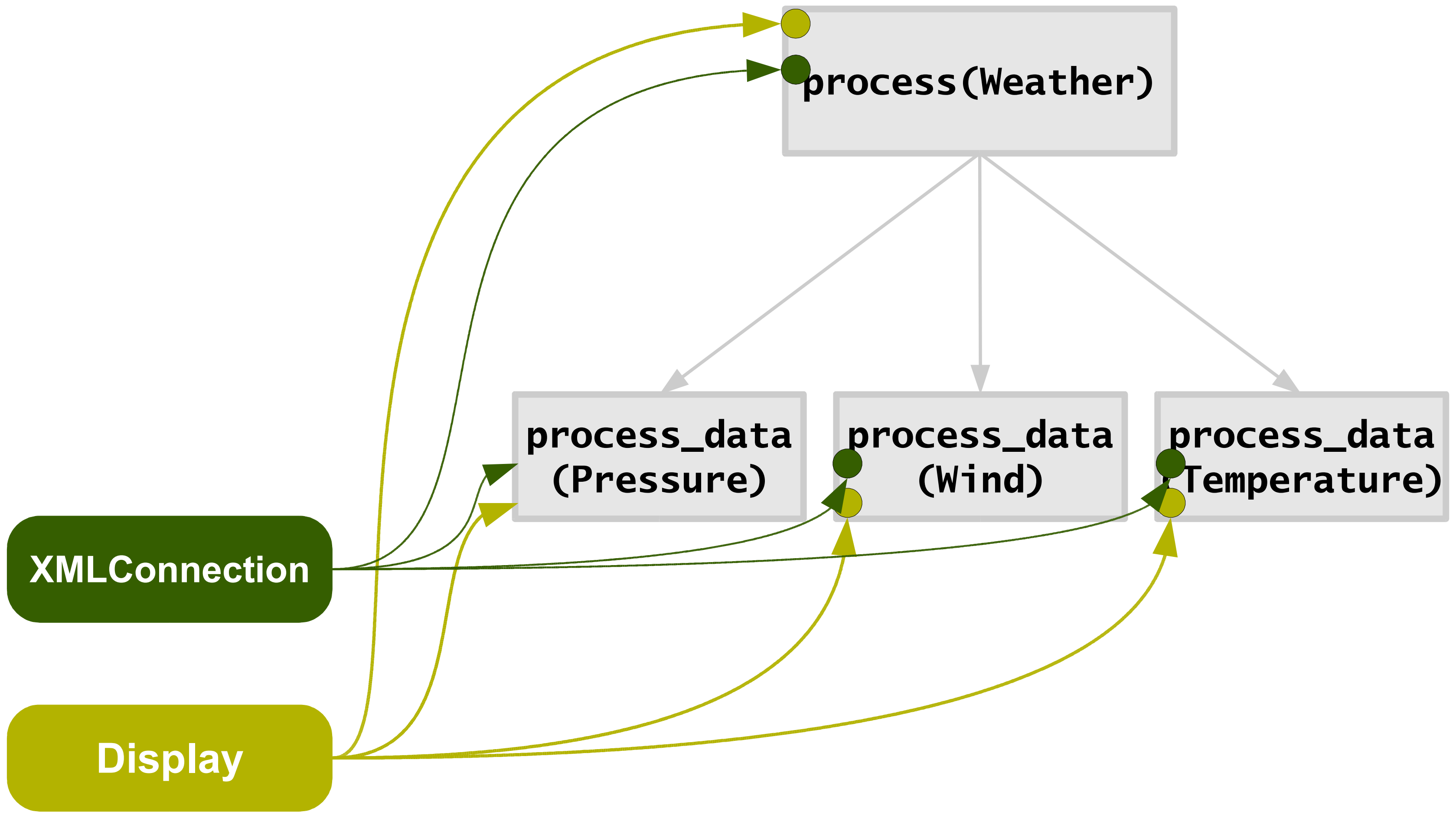
AspectC++

...at work

Actor Integration



Actor Integration



AspectC++

...at work

AspectC++ Join-Point API

Compile-Time Joinpoint API

JoinPoint::Result	Type of the function result
JoinPoint::Arg< ***i*** >::Type	Type of the ***i*** ***th*** function argument
JoinPoint::Arg< ***i*** >::ReferredType	(with 0 ≤ i < ARGS)
JoinPoint::ARGS	Number of arguments

...

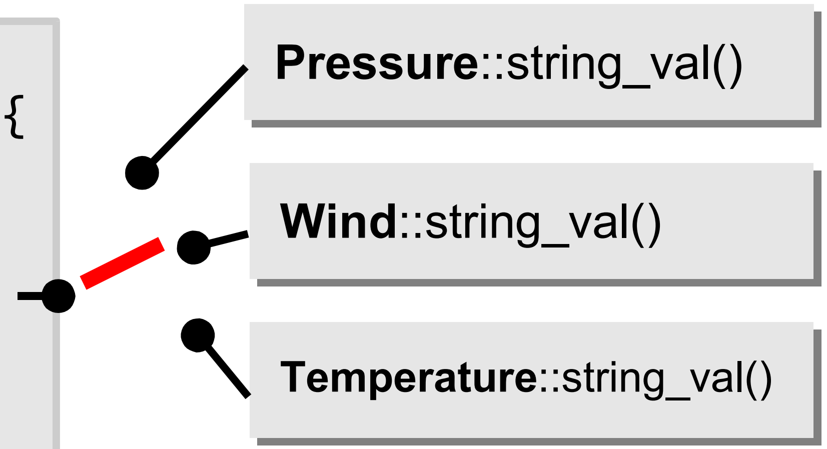
Runtime Joinpoint API

Result* result()	result value
Arg< ***i*** >::ReferredType* arg< ***i*** >()	value of ***i*** ***th*** argument

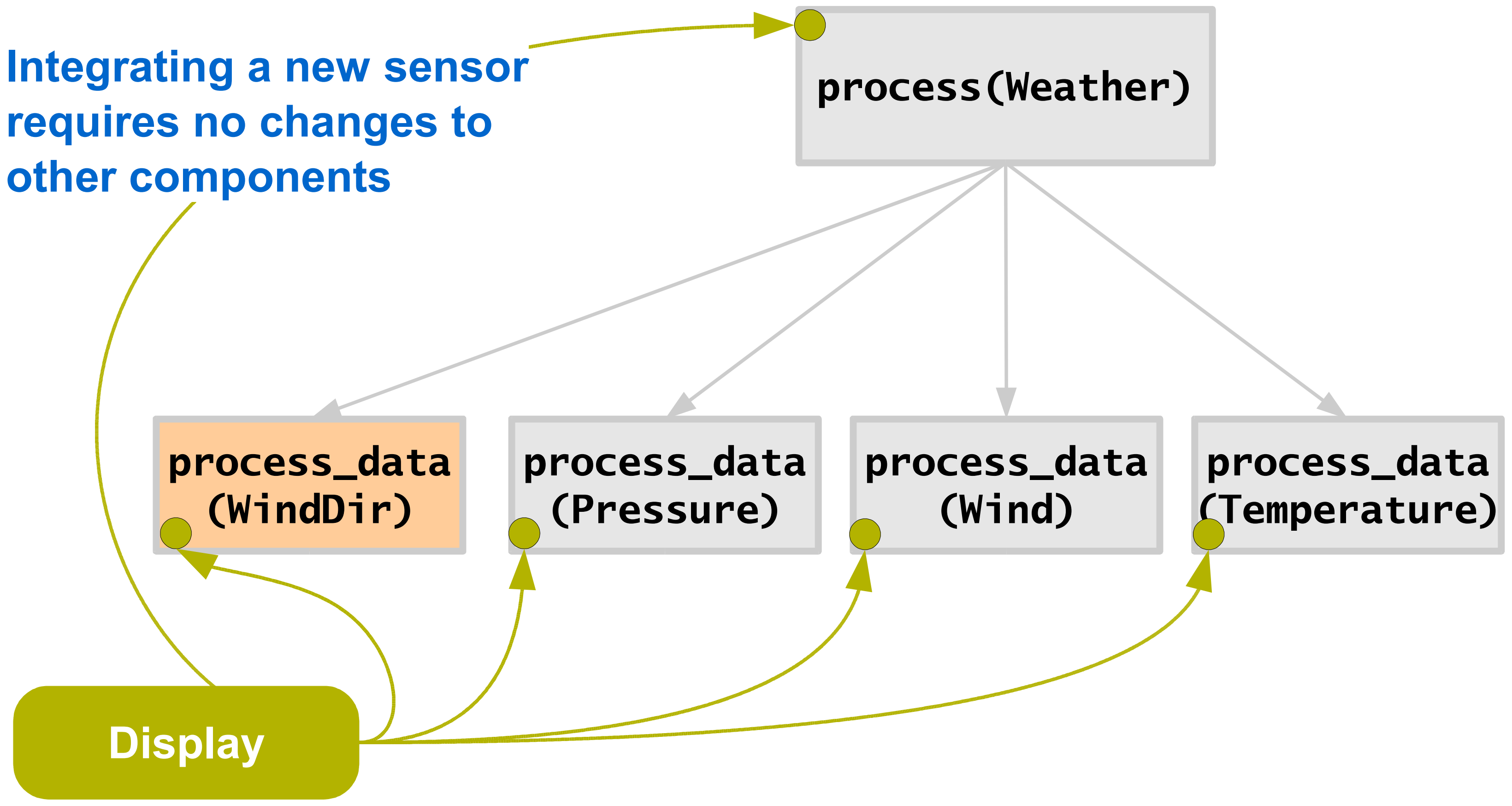
...

Sensors/Actors Connection with Generic Advice

```
advice execution("void
  Sink::process_data(%)") : before() {
  ...
  String<4> val_str;
  tjp->arg<0>()->string_val(val_str);
  ...
}
```

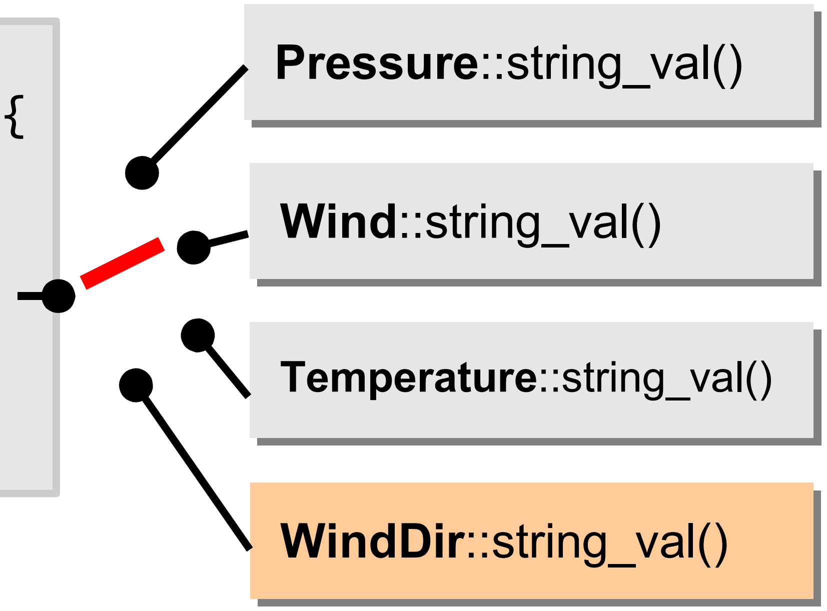


Sensors/Actors Connection with Generic Advice


Integrating a new sensor requires no changes to other components
process(Weather)
process_data (WindDir)
process_data (Pressure)
process_data (Wind)
process_data (Temperature)
Display

Sensors/Actors Connection with Generic Advice

```
advice execution("void
  Sink::process_data(%)") : before() {
  ...
  String<4> val_str;
  tjp->arg<0>()->string_val(val_str);
  ...
}
```



Design Conclusions

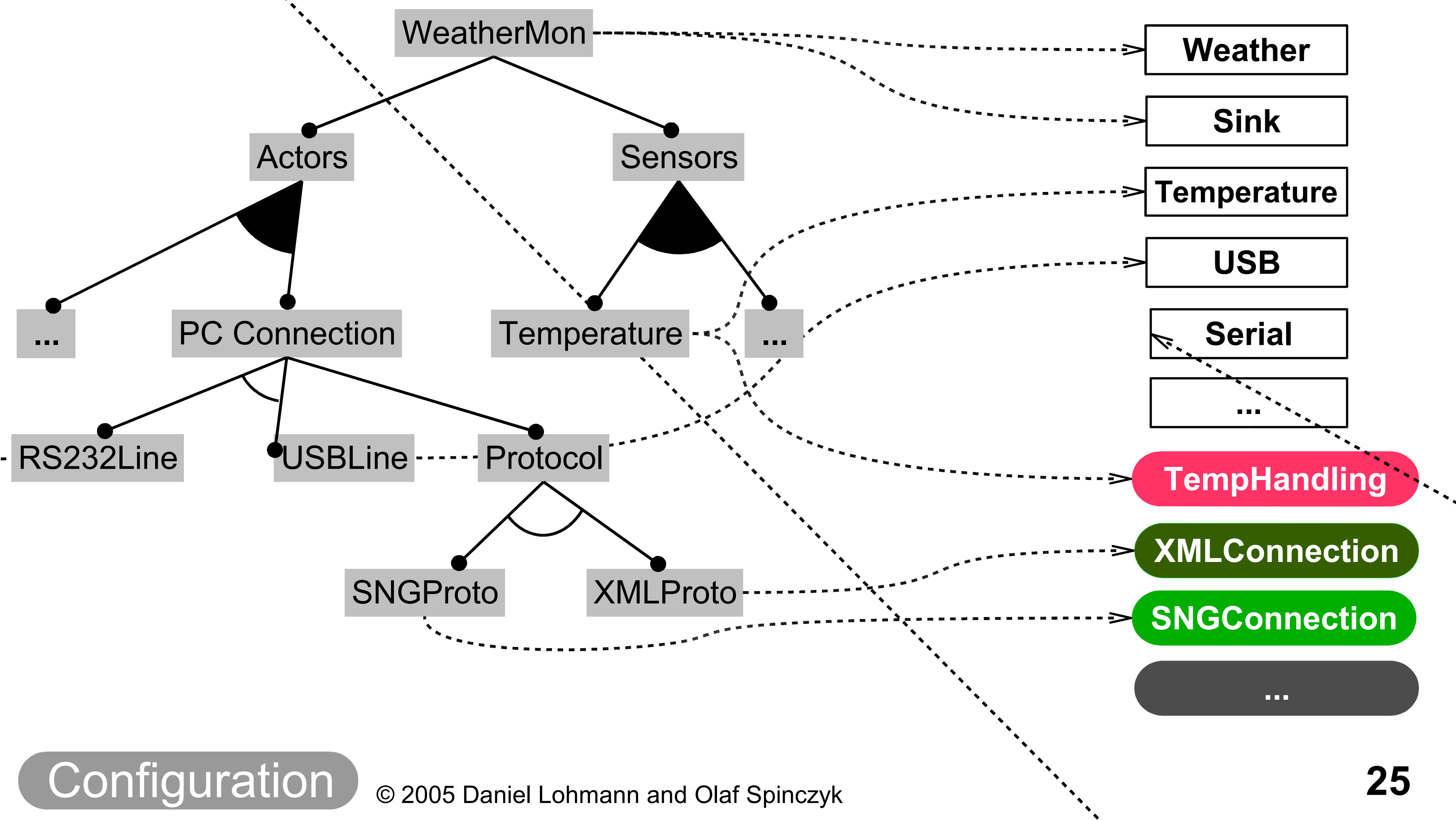
By using aspects, we achieved...

- complete decoupling of components
  - component slices are merged in by advice
  - actors and sensors “integrate themselves”
  - not a single #ifdef
- Plug & Play of components

...without sacrifying efficiency

- minimal stack usage due to advice code inlining
- everything is resolved at compile-time
  - no dynamic data structures to manage sensors/actors
  - no virtual functions

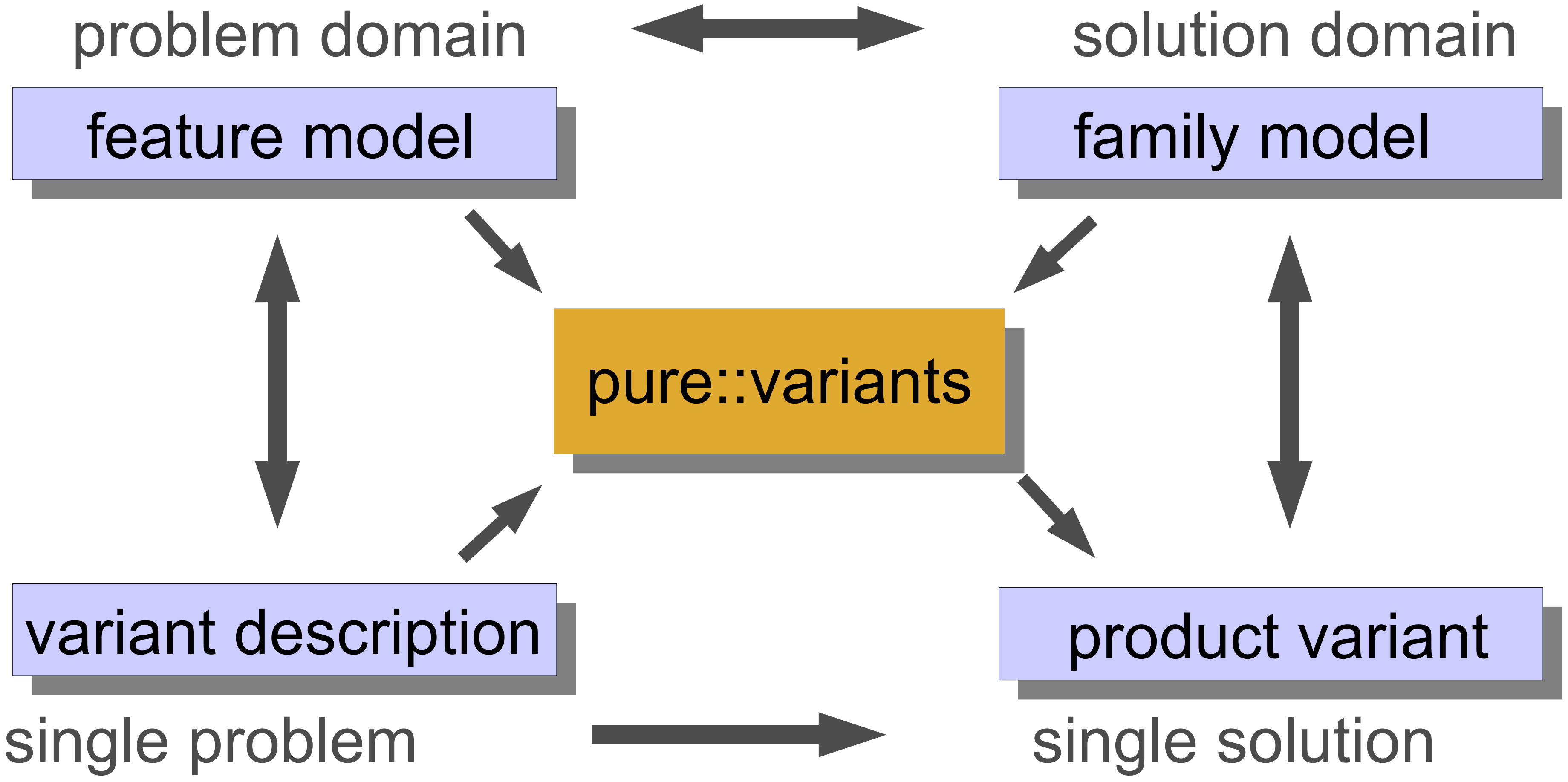
Configuration



pure::variants

- General-purpose tool for product-line engineering
  - based on program families and feature modeling
  - not restricted to AOP or AspectC++
  - but provides some special support for aspects
- Implemented as an Eclipse-plugin
- Commercial product from pure-systems GmbH
  - free “community edition” available
  - http://www.pure-systems.com

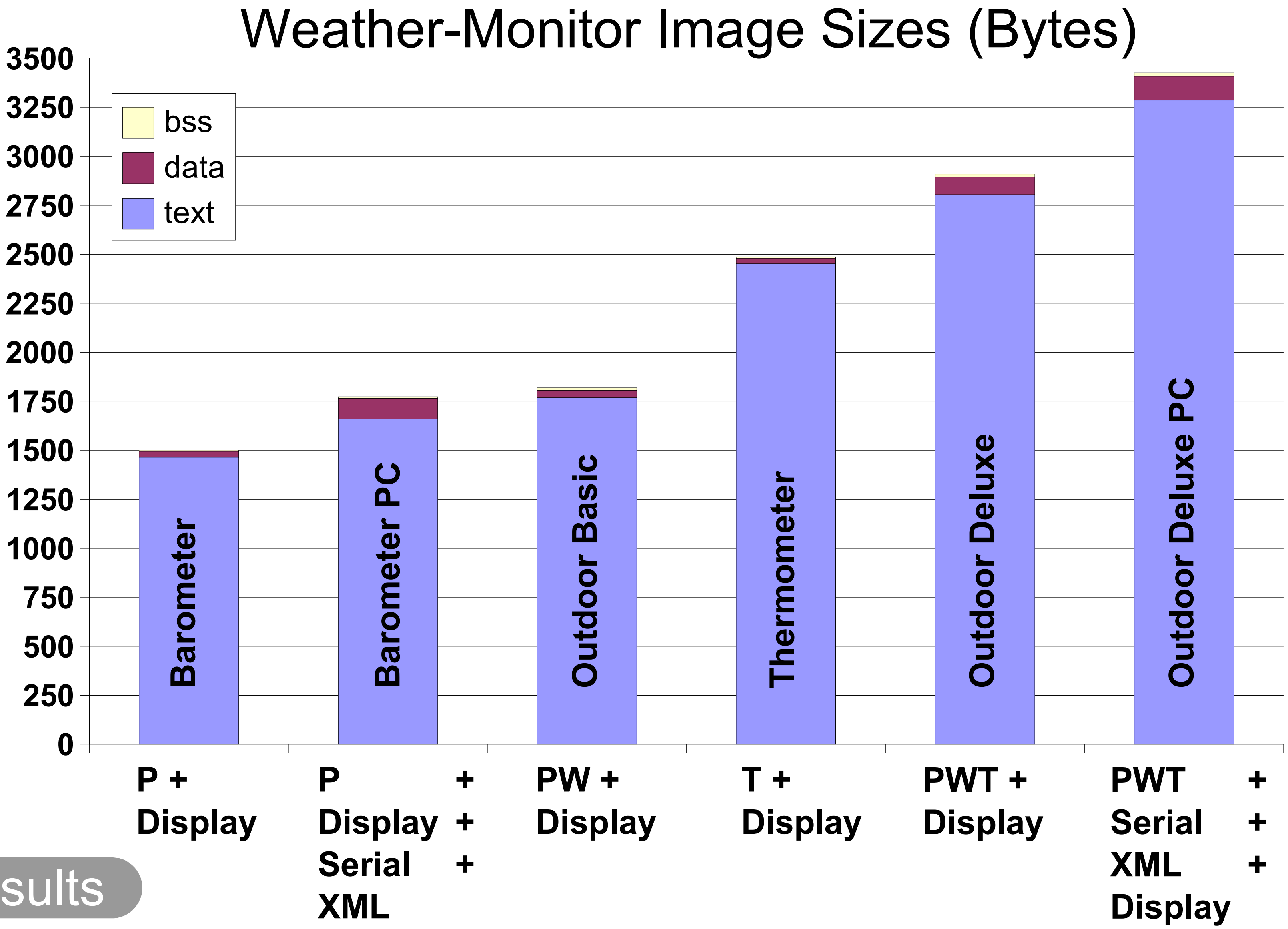
Variant Management



pure::variants

...at work

Scalability of the Product Line



Results

Summary: This Demo showed..

- **Aspects in embedded product lines**
  - loose coupling of components
  - Plug&Play configurability
  - highly efficient code

AOP provides real benefits for product-line development!

- **Complete, Eclipse-based tool chain**
  - AspectC++ Eclipse plug-in (ACDT)
  - pure::variants

All required tools are available today!

Questions?

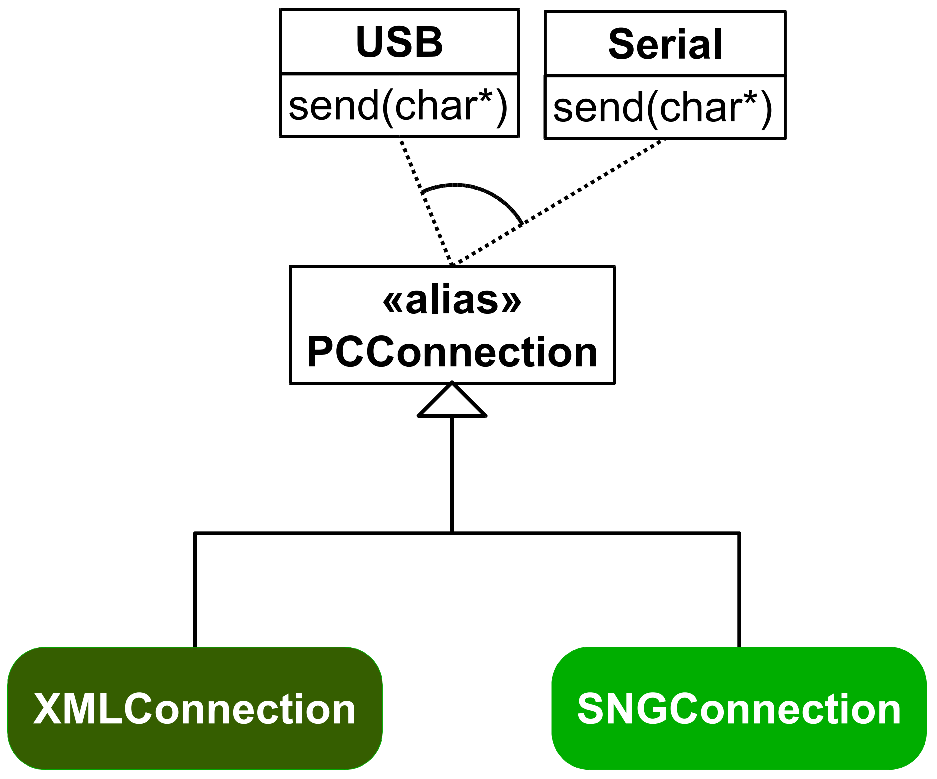
Download AspectC++ from

www.aspectc.org

More about pure::variants at

www.pure-systems.com

PC Connection Integration


USB
send(char*)
Serial
send(char*)
«alias»
PCConnection
XMLConnection
SNGConnection