
NS
18
NIAGARA
SUMMIT
CONNECTING
THE WORLD

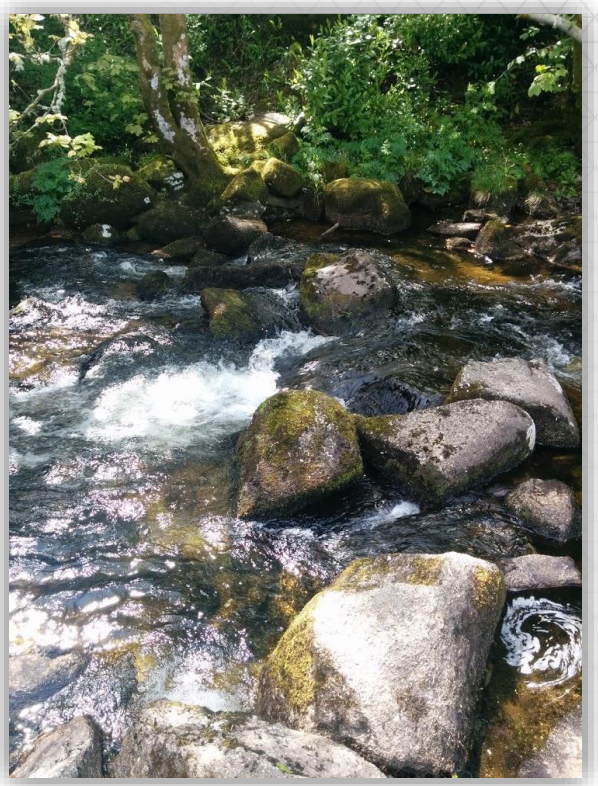
Common Niagara Development Pitfalls

Gareth Johnson, Niagara Core Architect

What is a pitfall?

An easy to make mistake that can lead to…

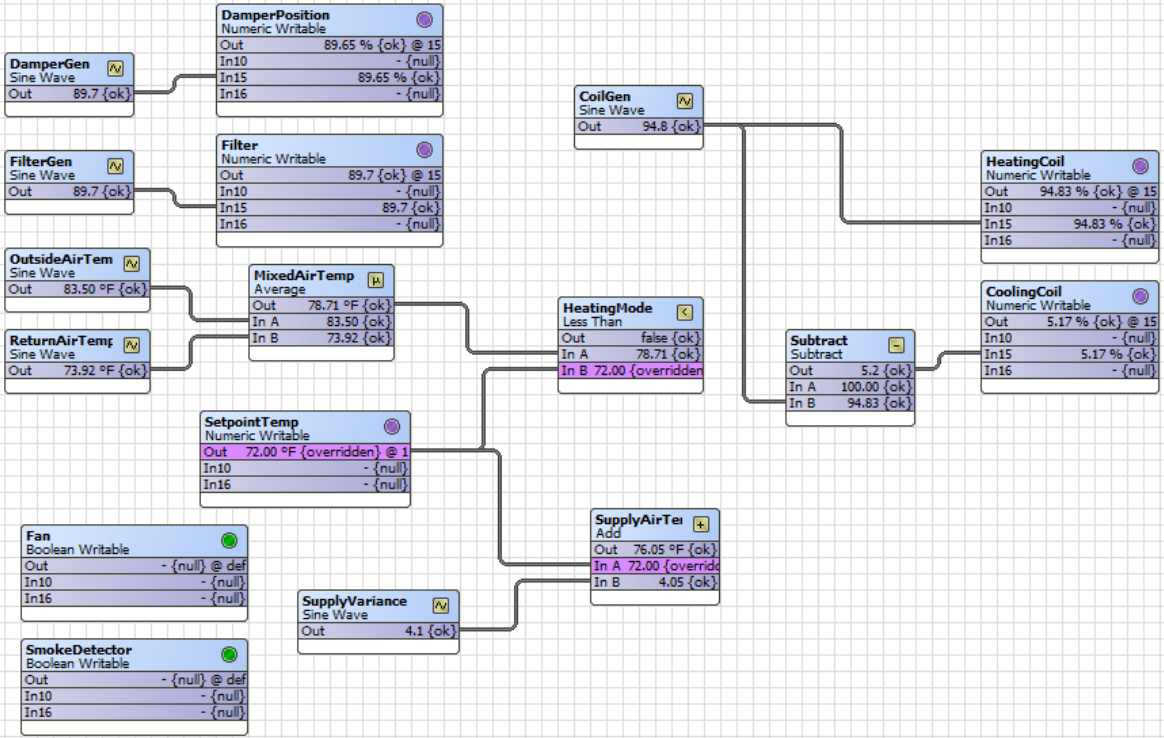
- Unintended or unpredictable behavior
- A security vulnerability
- A bug



Overview

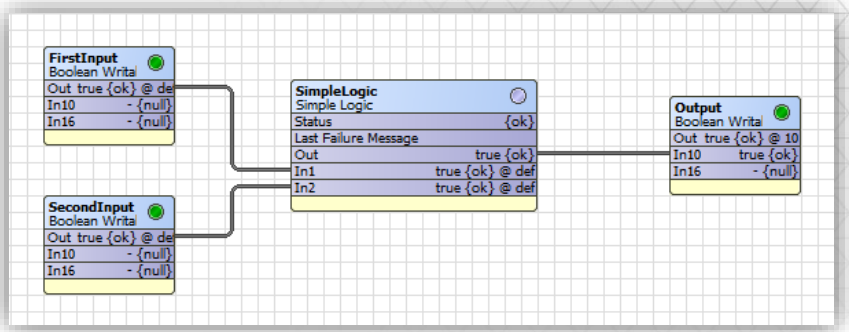
- Component Model
- Server side
- Client side

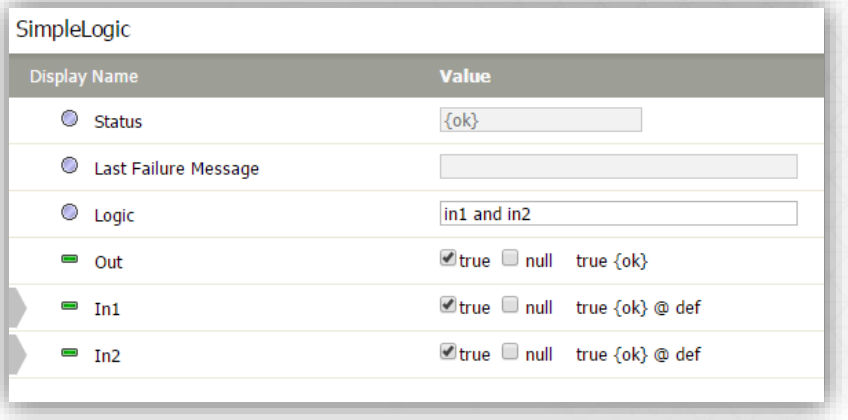
Component Model Pitfalls


DamperPosition
Numeric Writable
Out 89.65 % {ok} @ 15
In10 - {null}
In15 89.65 % {ok}
In16 - {null}
DamperGen
Sine Wave
Out 89.7 {ok}
Filter
Numeric Writable
Out 89.7 {ok} @ 15
In10 - {null}
In15 89.7 {ok}
In16 - {null}
FilterGen
Sine Wave
Out 89.7 {ok}
OutsideAirTem
Sine Wave
Out 83.50 °F {ok}
MixedAirTemp
Average
Out 78.71 °F {ok}
In A 83.50 {ok}
In B 73.92 {ok}
ReturnAirTemp
Sine Wave
Out 73.92 °F {ok}
CoilGen
Sine Wave
Out 94.8 {ok}
HeatingCoil
Numeric Writable
Out 94.83 % {ok} @ 15
In10 - {null}
In15 94.83 % {ok}
In16 - {null}
HeatingMode
Less Than
Out false {ok}
In A 78.71 {ok}
In B 72.00 {overridden
Subtract
Subtract
Out 5.2 {ok}
In A 100.00 {ok}
In B 94.83 {ok}
CoolingCoil
Numeric Writable
Out 5.17 % {ok} @ 15
In10 - {null}
In15 5.17 % {ok}
In16 - {null}
SetpointTemp
Numeric Writable
Out 72.00 °F {overridden} @ 1
In10 - {null}
In16 - {null}
SupplyAirTe
Add
Out 76.05 °F {ok}
In A 72.00 {overrid
In B 4.05 {ok}
Fan
Boolean Writable
Out - {null} @ def
In10 - {null}
In16 - {null}
SupplyVariance
Sine Wave
Out 4.1 {ok}
SmokeDetector
Boolean Writable
Out - {null} @ def
In10 - {null}
In16 - {null}

BSimpleLogic

- Has two inputs and an output
- Has a ‘logic’ string that’s processed against inputs to create the output
- Compiles logic string for optimum performance.





```
public class BSimpleLogic extends BComponent {
  ...
  @Override
  public void stationStarted() {
    build();
    execute();
  }

  @Override
  public void changed(Property property, Context context) {
    if (property == logic) {
      build();
      execute();
    }
    else if (property == in1 || property == in2) {
      execute();
    }
  }

  private void build() { ... }
  private void execute() { ... }
}
```

Pitfall #1: component started callbacks

- stationStarted() will only be called when the Station starts.
- stationStarted() won't be called when the Component is first added.
- Use started() to handle both scenarios.

```
public class BSimpleLogic extends
BComponent {
  ...
  @Override
  public void started() {
    build();
    execute();
  }
}
```

Pitfall #2: property equality

- If another class extends BSimpleLogic, the properties can be overridden.
- The changed callback will no longer work.
- Always use `equals(…)` instead of `==`

```
@Override
public void changed(Property prop,
                    Context cx) {
  if (prop.equals(logic)) {
    build();
    execute();
  }
  else if (prop.equals(in1) ||
           prop.equals(in2)) {
    execute();
  }
}
```

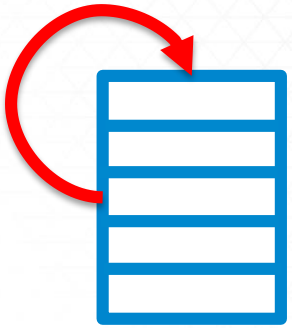
Pitfall #3: components in different environments

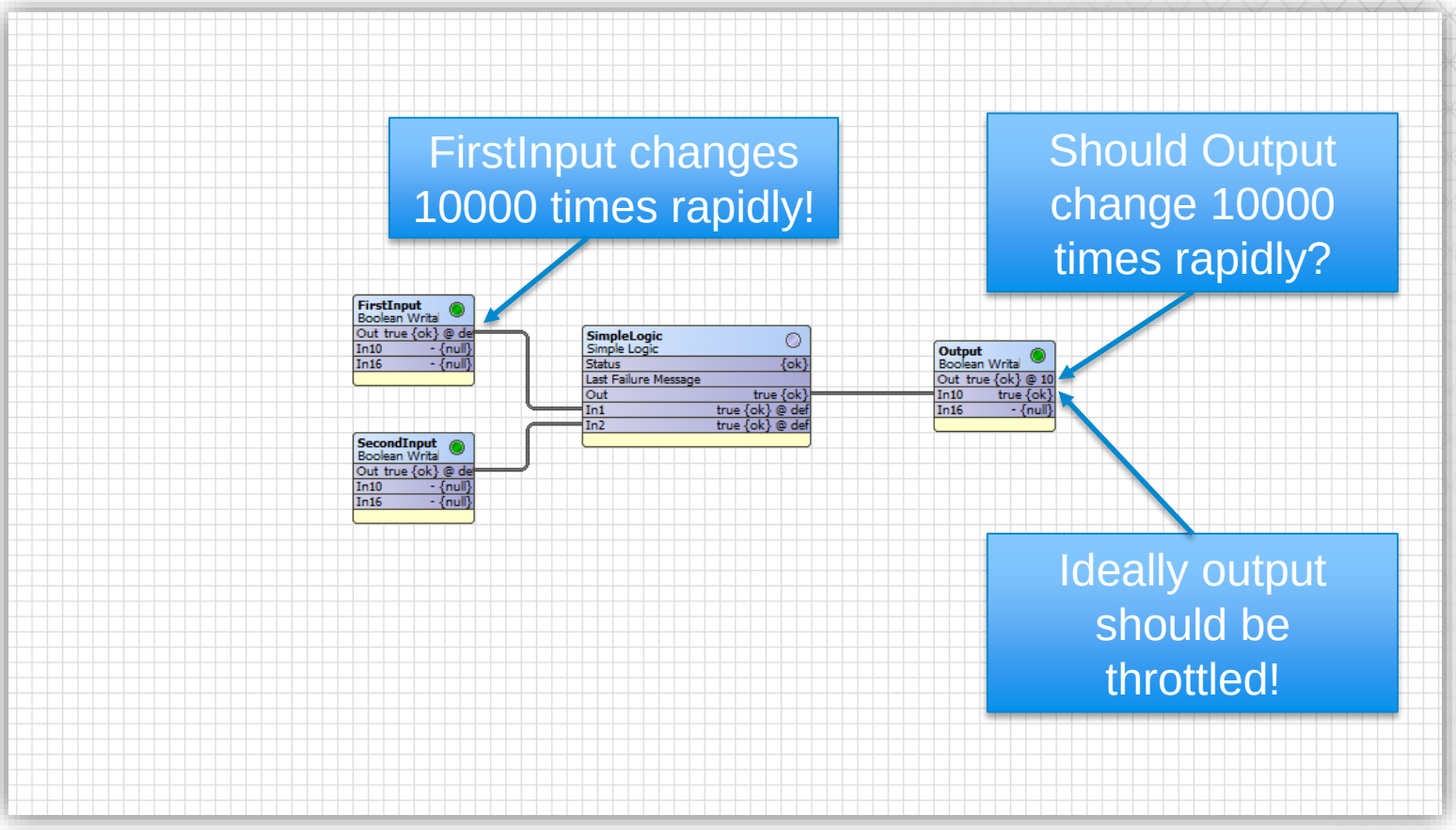
- Components can run in different environments…
  - Station
  - Workbench
  - BOG files
- Changed callbacks will be made even in a Workbench environment!
- Always use `isRunning()` checks to ensure code is running in a Station's Component Space

```
@Override
public void changed(Property prop,
                    Context cx) {
  if (isRunning()) {
    if (prop.equals(logic)) {
      build();
      execute();
    }
    else if (prop.equals(in1) ||
      prop.equals(in2)) {
      execute();
    }
  }
}
```

Niagara Engine Thread

- Main control engine thread!
- Constantly loops…
  - Runs default async Action invocations
  - Checks for system clock changes
  - Checks and runs timers



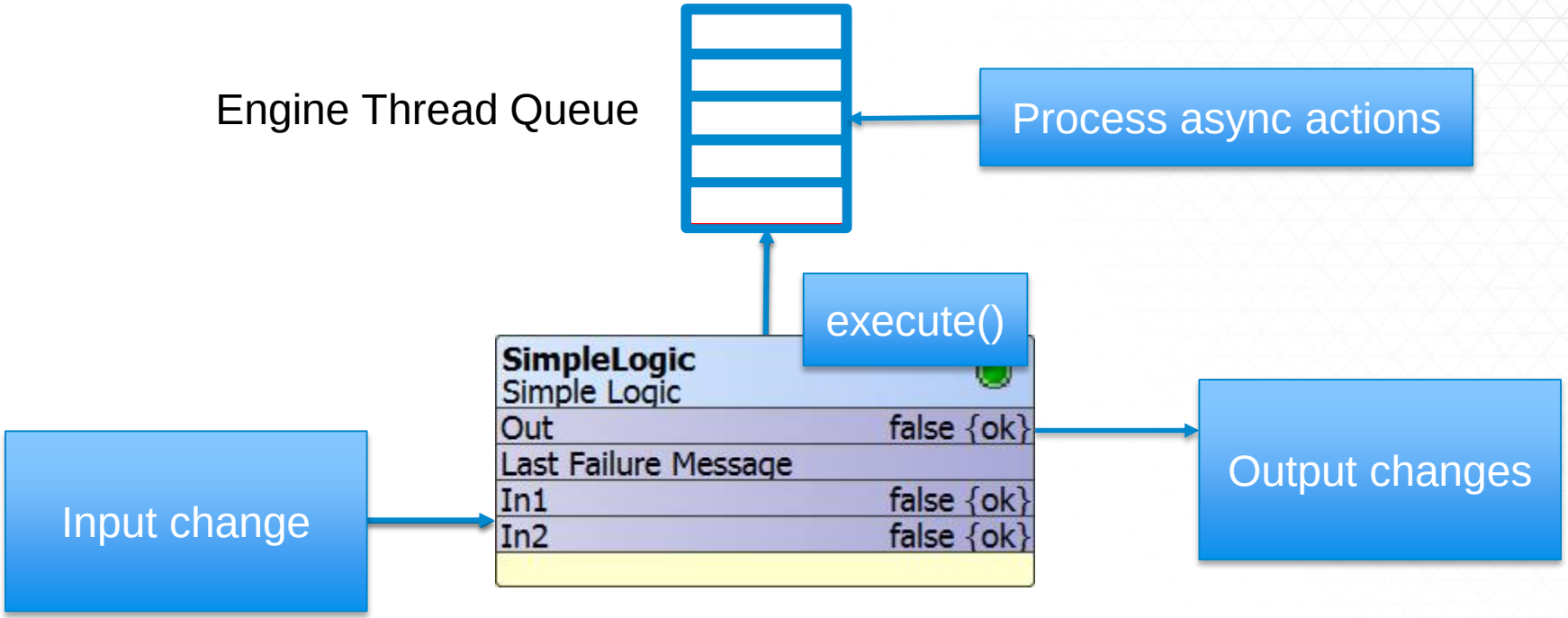

FirstInput changes 10000 times rapidly!
Should Output change 10000 times rapidly?
FirstInput
Boolean Writa
Out true {ok} @ de
In10 - {null}
In16 - {null}
SecondInput
Boolean Writa
Out true {ok} @ de
In10 - {null}
In16 - {null}
SimpleLogic
Simple Logic
Status {ok}
Last Failure Message
Out true {ok}
In1 true {ok} @ def
In2 true {ok} @ def
Output
Boolean Writa
Out true {ok} @ 10
In10 true {ok}
In16 - {null}
Ideally output should be throttled!

Pitfall #4: component efficiency

- Actions marked with `async` slot flag
- By default, async Actions run in Engine Thread
- Default async Actions are coalesced.
- Can override BComponent#post(…) method to run in a different thread

```
...
@NiagaraAction(
  name = "execute",
  flags = Flags.ASYNC |
Flags.HIDDEN)
public class BSimpleLogic extends
BComponent {
  ...
  public void doExecute() {…}
}
```

Asynchronous Actions in action!!!



```
@NiagaraAction(name="execute", flags = Flags.ASYNC | Flags.HIDDEN)
public class BSimpleLogic extends BComponent {
  …
  @Override
  public void changed(Property prop, Context cx) {
    if (isRunning()) {
      if (prop.equals(logic)) {
        build();
        execute();
      }
      else if (prop.equals(in1) || prop.equals(in2)) {
        execute();
      }
    }
  }
  private void build() { ... }
}
```

Pitfall #5: callbacks and threading

- BComponent callbacks can be called from different threads…
  - Engine thread
  - Fox session threads
  - Web server (BOX) threads
- Use synchronization on member variables
- Force all updates on member variables to happen in Engine thread using default async Actions!

```
public class BSimpleLogic extends BComponent {
  …
  @Override
  public void changed(Property prop, Context cx) {
    if (isRunning() && prop.equals(logic)) {
      build();
    }
  }

  void build() {
     …
     expr = ExpressionFactory.compile(getLogic());
     …
  }

  volatile Expression expr = NullExpression.INSTANCE;
}
```

Refactor – extend from BBooleanPoint

- BControlPoint architecture implements async execute pattern
- BBooleanPoint implements BIBoolean, BIEnum, BIStatusValue interfaces
  - Used in graphics!
- Point extensions can be added for alarming, histories etc

```
public class BSimpleLogic
  extends BBooleanPoint {

  @Override
  public void onExecute(BStatusValue out,
                        Context cx) {
    …
     BStatusBoolean bout = (BStatusBoolean)out;
     bout.setValue(expr.execute(model));
  }
}
```

Pitfall #6 blocking the Engine thread!

- Engine thread is designed for processing lots of small non-blocking operations.
- Long running or blocking operations can cause an Engine watchdog timeout!
- Things that shouldn't run on the Engine thread…
  - Any blocking serial, network or disk IO calls.
  - Long running queries (i.e. BQL)
  - Any code that can block (i.e. heavily multi-threaded code).

Tips for threading

- Any blocking operations should run on a separate thread!
- Try not to persist long running background threads
  - More context switching
  - More stack space is consumed
- Threads should always have a name!
- Consider using javax.baja.util.ExecutorUtil
  - Methods for starting short term background threads…
    - newSingleThreadBackgroundExecutor(…)
  - Has method for shutting down executors…
    - shutdown(…)

Pitfall #7: unoptimized slot flags

- To avoid persisting data unnecessarily…
  - Use the transient slot flag
- To avoid filling up data recovery service…
  - Use the nonCritical or transient slot flags
- To avoid filling up the audit log unnecessarily…
  - Use the noAudit slot flag

Server side Pitfalls

Pitfall #8: not passing the Context

- Context can contain…
  - BUser – used to enforce Niagara's security model
  - Locale – used to resolve the correct lexicon
- If you're passed a 'Context' then pass it on. Especially when…
  - Resolving an ORD
    - Enforces security model
  - Interacting with the Component Model
    - Enforces security model
    - Creates audit trail of changes made

Context from an Action handler

```
@NiagaraAction(
  name="override",
  parameterType="control:NumericOverride",
  defaultValue="new BNumericOverride()"
)
public class BMyComponent extends BComponent {
  …
  @Override
  public void doOverride(BNumericOverride override) {
    …
  }
}
```

Context from an RPC call

```
@NiagaraRpc(
  transports = { @Transport(type=box) }
)
public Map<String, String> foo(String param, Context cx) {
  setBoolean(someProperty, true, cx);
}
```

Context from an HTTP Servlet

```
public class MyServlet extends HttpServlet {
  protected void doGet(HttpServletRequest req,
                       HttpServletResponse resp) {
    Context cx = (Context)req.getAttribute("niagara.context");
    BFolder folder = (BFolder)BOrd.make("station:|slot:/Folder")
      .get(cx);
    …
  }
}
```

BUser.getCurrentAuthenticatedUser()

- Returns currently authenticated user
- Useful for security sensitive pieces of code
- Not reliant on Context

Client side Pitfalls

Pitfall #9: not importing BajaScript Types

- Since BajaScript v2 **all** network calls are asynchronous
- Creating instances Types must have a corresponding import

```
define(['baja!',
        'baja!control:NumericWritable'],
       function (baja, types) {
  return function foo() {
    var nw =
      baja.$('control:NumericWritable');
  };
});
```

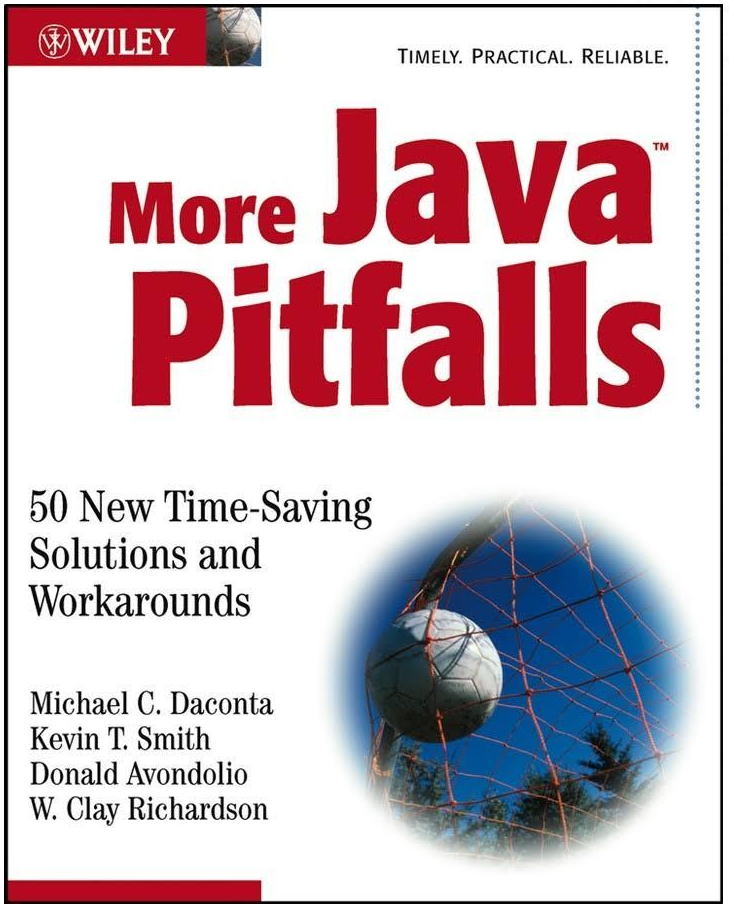
Pitfall #10: CSS clashes

- bajaux Widgets all share common DOM in Single Page Application
- CSS selectors must be unique and should not clash with each other
- Try using a standard convention to ensure uniqueness…
  - company-module-widget

```
.acme-mymodule-chart-panel {
  background-color: red;
  color: blue;
  padding: 1em;
}
```

Conclusion

- Most pitfalls are really about creating software that runs well within the Niagara framework!
- To try and avoid pitfalls ensure you're using a great automated test and code review strategy!


WILEY
TIMELY. PRACTICAL. RELIABLE.
More Java™ Pitfalls
50 New Time-Saving Solutions and Workarounds
Michael C. Daconta
Kevin T. Smith
Donald Avondolio
W. Clay Richardson