SML: A New Interface Into Soar

Douglas Pearson
+ Bob, Jon, Karen, Trevor, Devvan, John

douglas.pearson@threepenny.net

Short-Term Problem

- New debugger
  - Avoid string parsing
  - Use Java for faster tool development
  - Dynamic attachment to existing processes
  - High performance
- gSKI integration
  - Removal of existing command line
- Redux choice to use JESS for rule matcher

Long-Term Problem

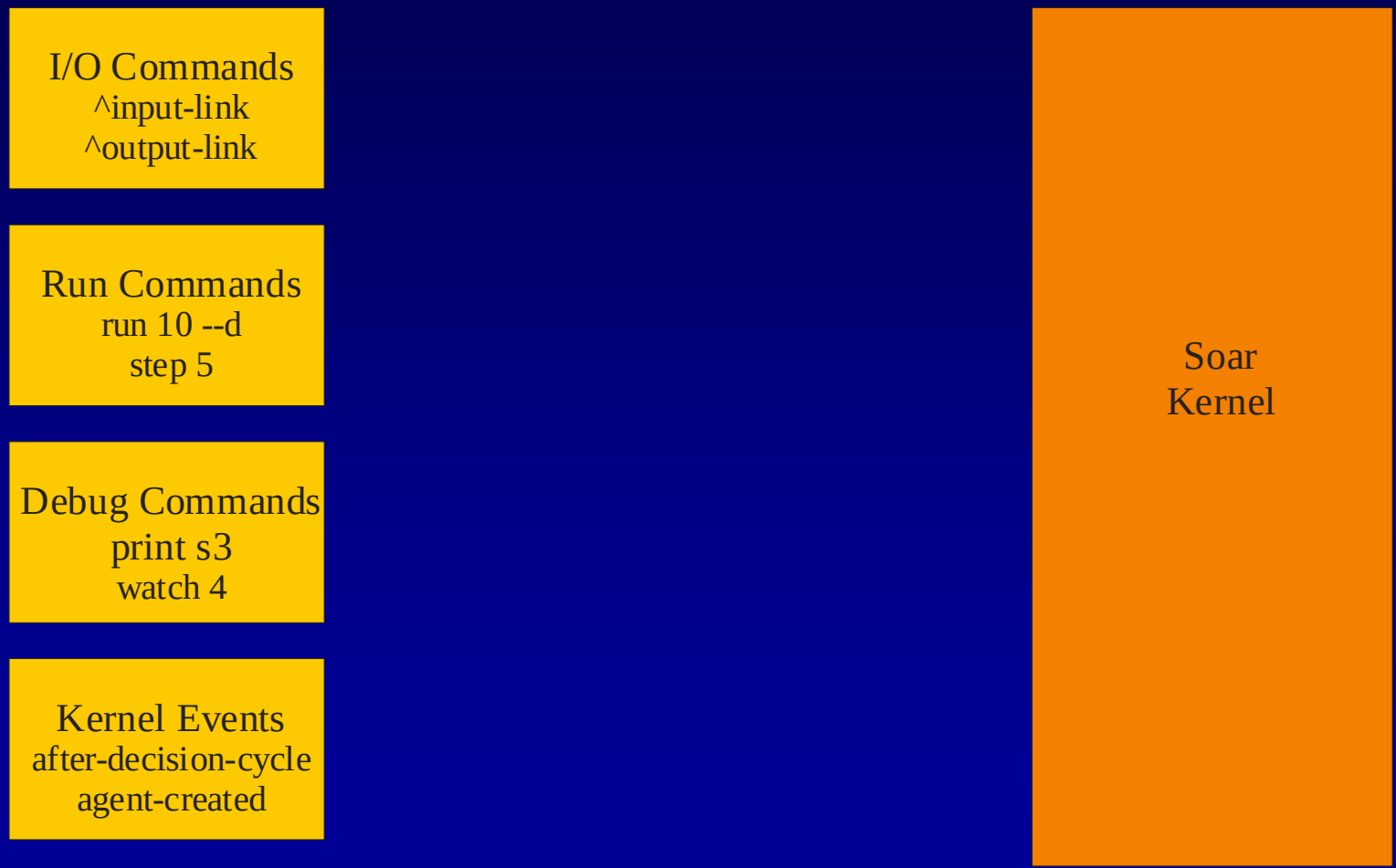
- Interfacing to Soar has always been hard
  - Initially limited to implementation language (LISP/C)
  - Extended to Tcl but lead to dependencies on Tcl
  - SGIO supported embedded kernels but only for I/O
  - gSKI added clean interface to kernel but multiple languages, debugging and remote I/O unimplemented.

Key Properties of our Solution

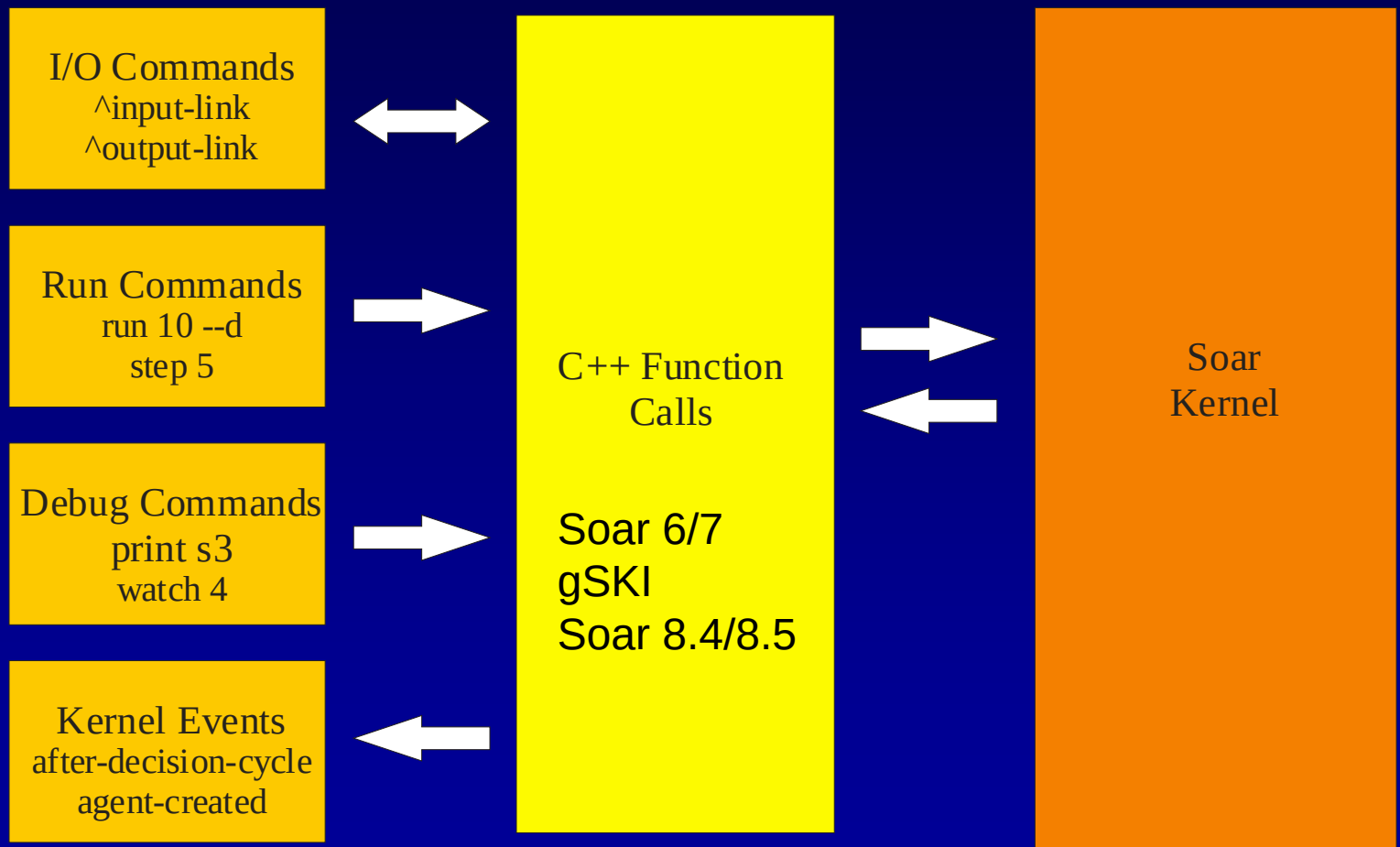
- Supports multiple languages (Java, C++, Tcl currently) while removing Tcl dependency
- Supports uniform interface for I/O (environments) and commands (debuggers)
- Supports embedding kernel within debugger or environment with remote connections between them
- Supports multiple clients (environments, tools, debuggers etc.) connecting to a single kernel
- Supports dynamic connection and disconnection of tools (esp. debuggers) from a running kernel
- Provides a uniform, high-level, data-driven model for the entire Soar interface while achieving high performance
- Moves command line support out of the kernel while providing universal access to it from any client
- Includes a new cleaned up command line interface

- Lots of new capabilities yet in most cases the new interface is substantially faster than 8.5.2
- No production level changes (except ‘tcl’ -> ‘exec’)

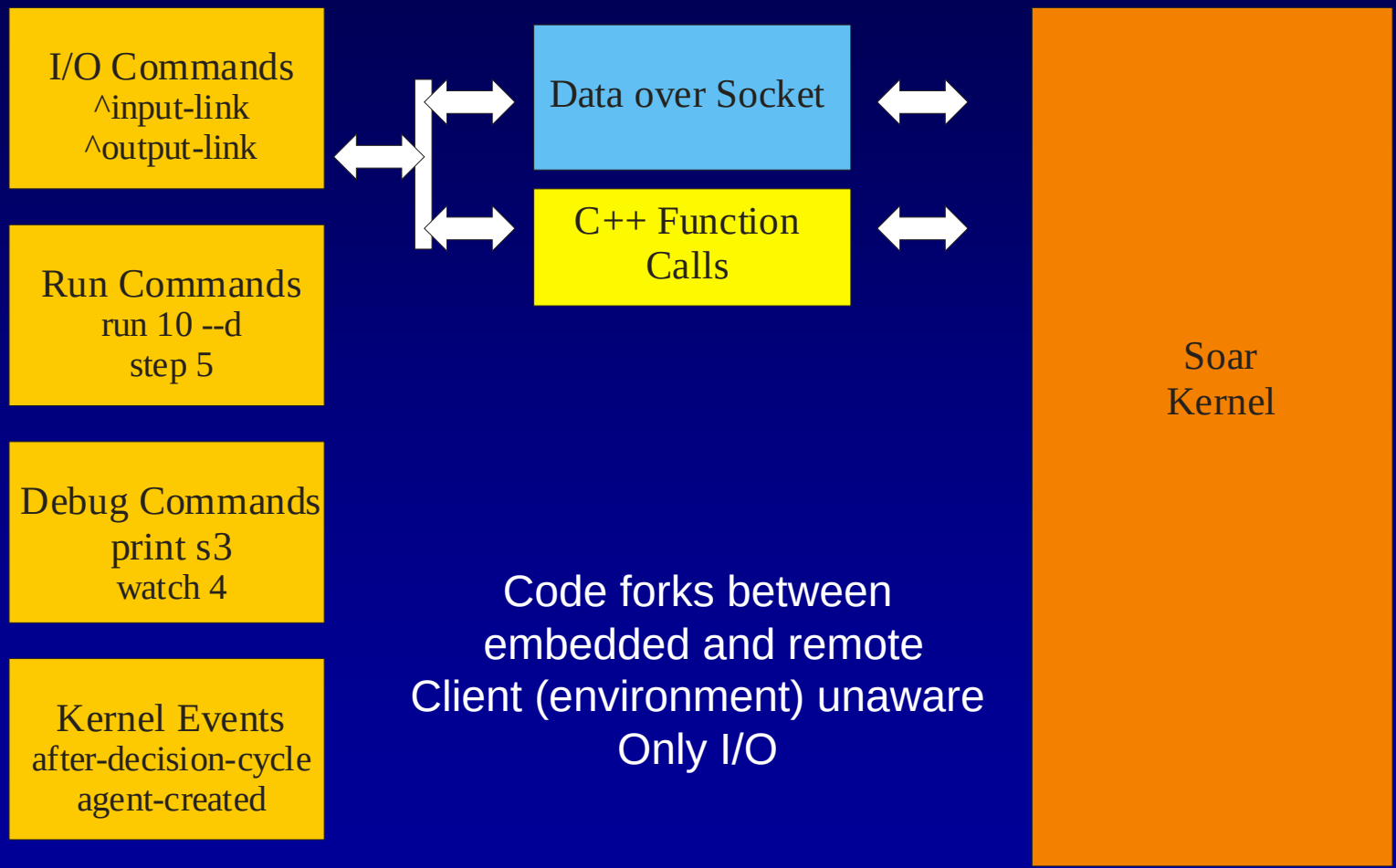
Connecting to Soar



Connecting to Soar
Direct Link to Kernel

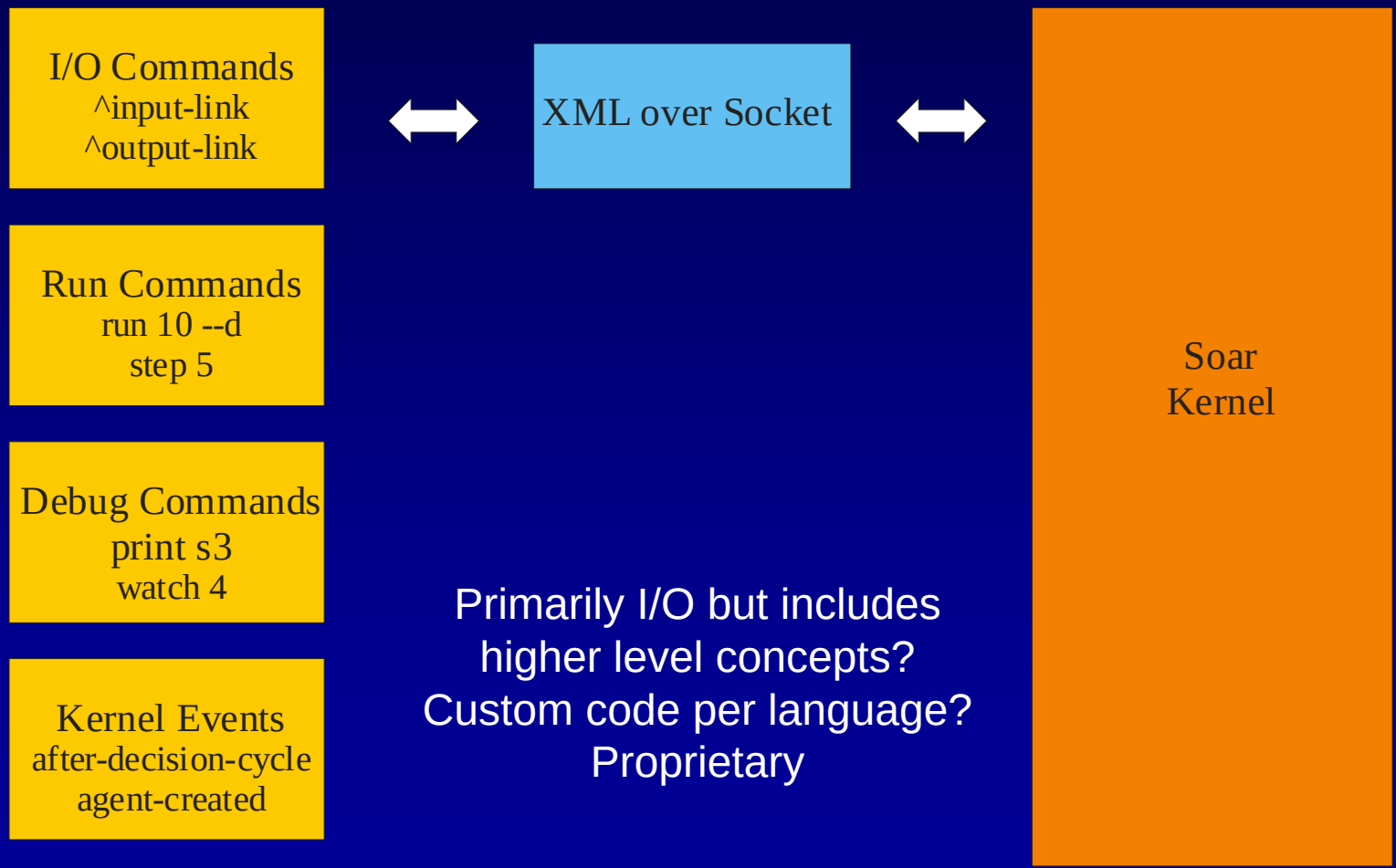


Connecting to Soar
SGIO Style



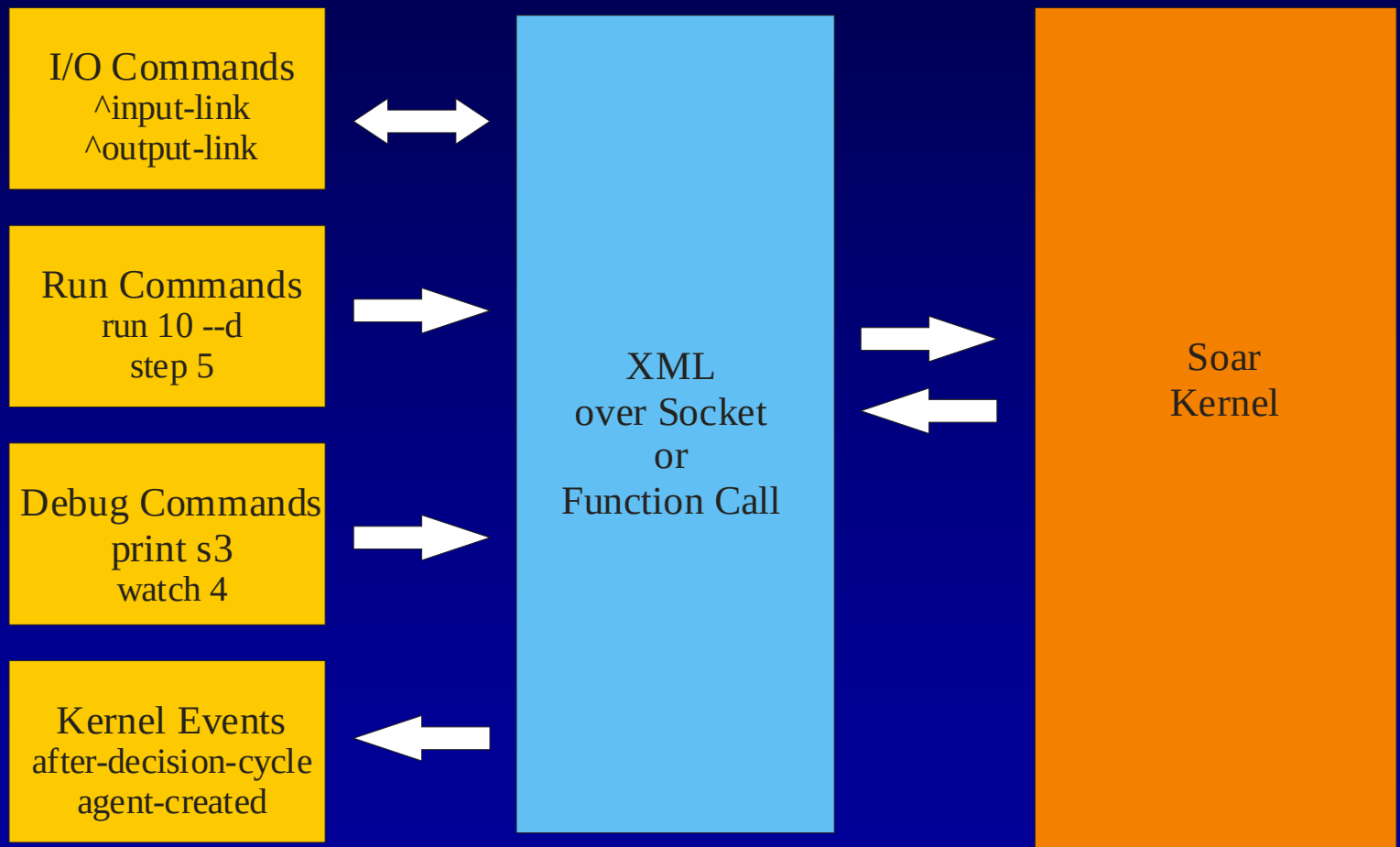
ThreePenny SOFTWARE

Connecting to Soar ATE Style?



ThreePenny SOFTWARE

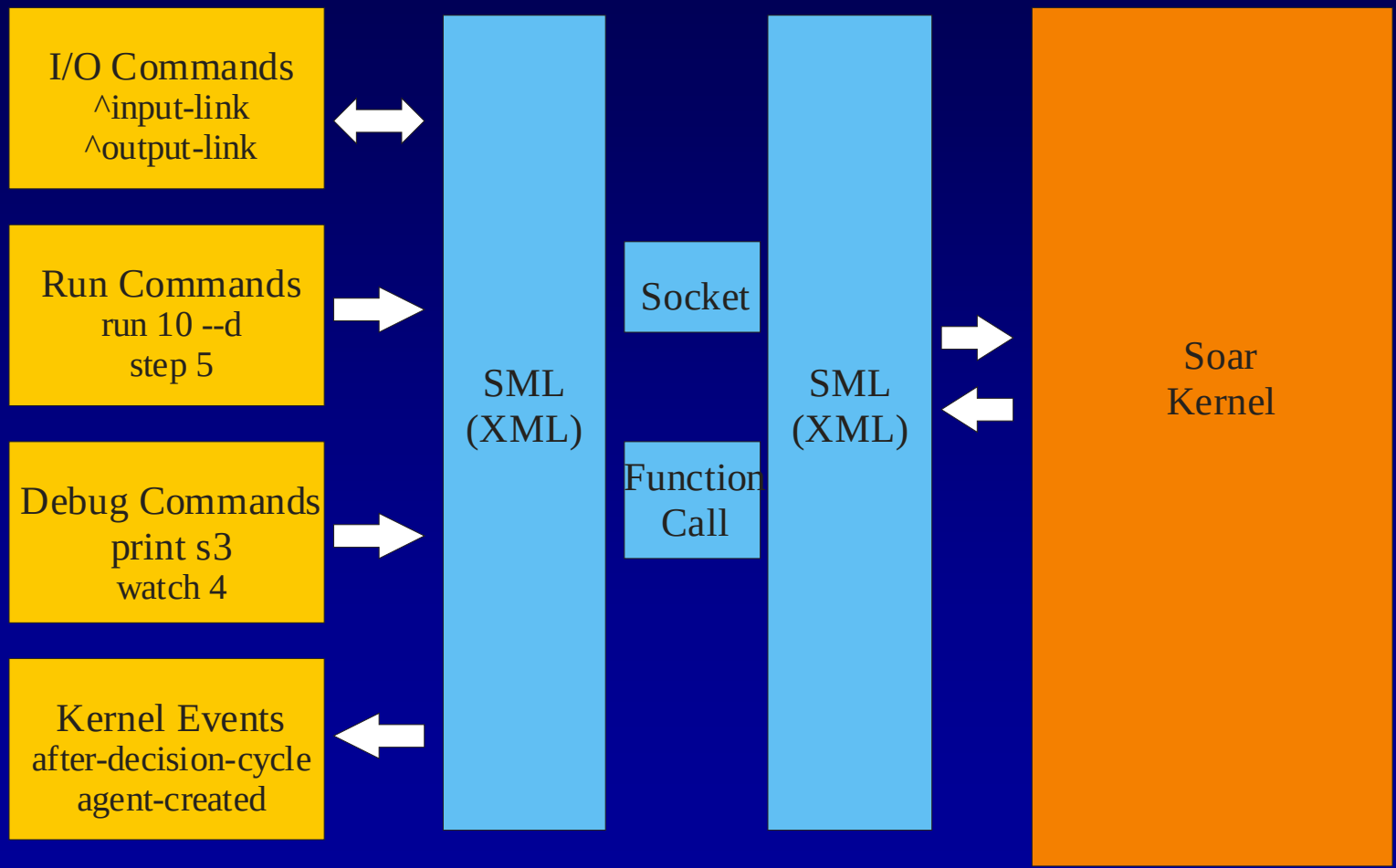
Connecting to Soar
SML Style – Soar Markup Language



Uniform for entire interface

ThreePenny SOFTWARE

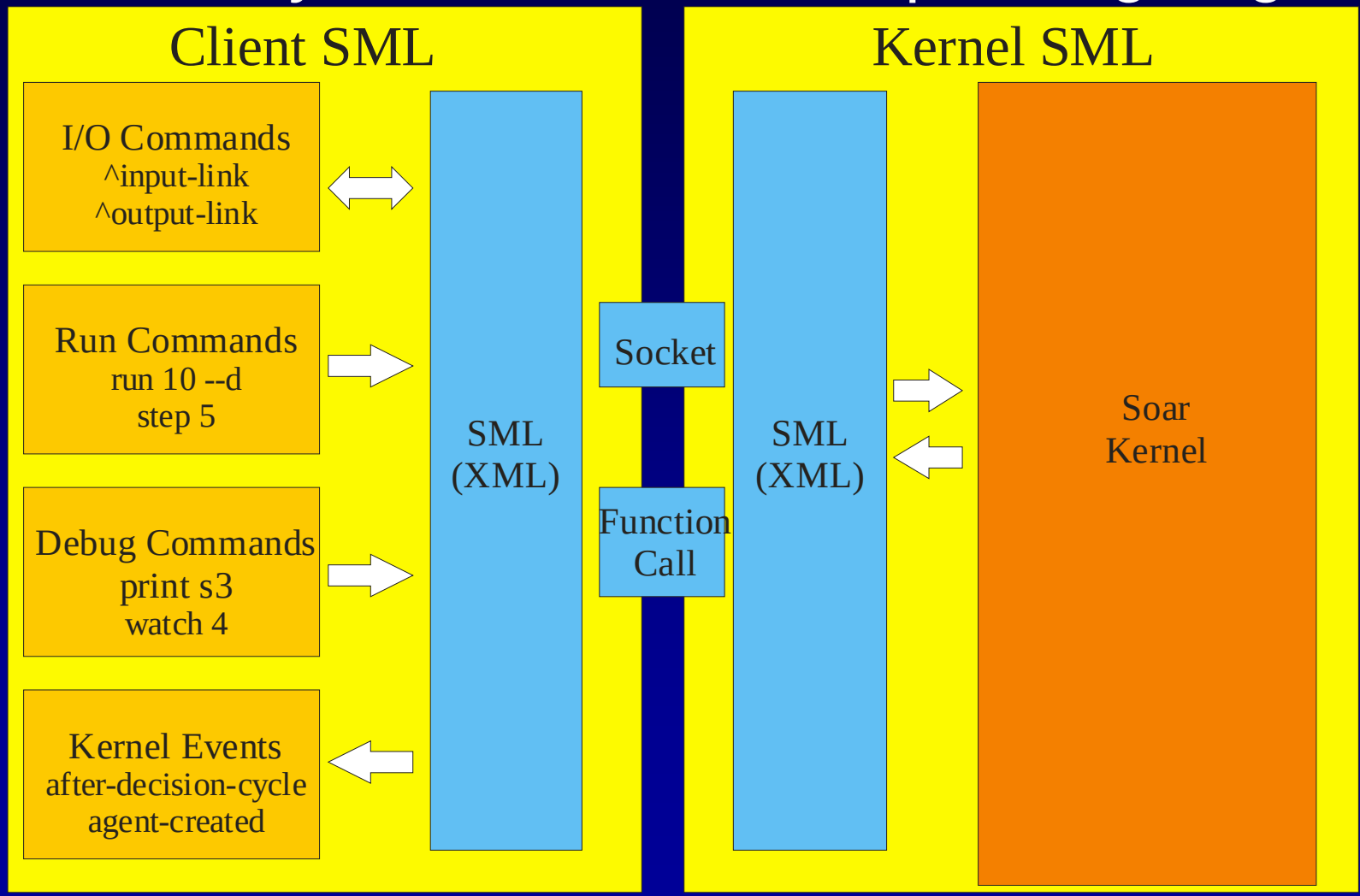
Connecting to Soar
SML Style – Soar Markup Language



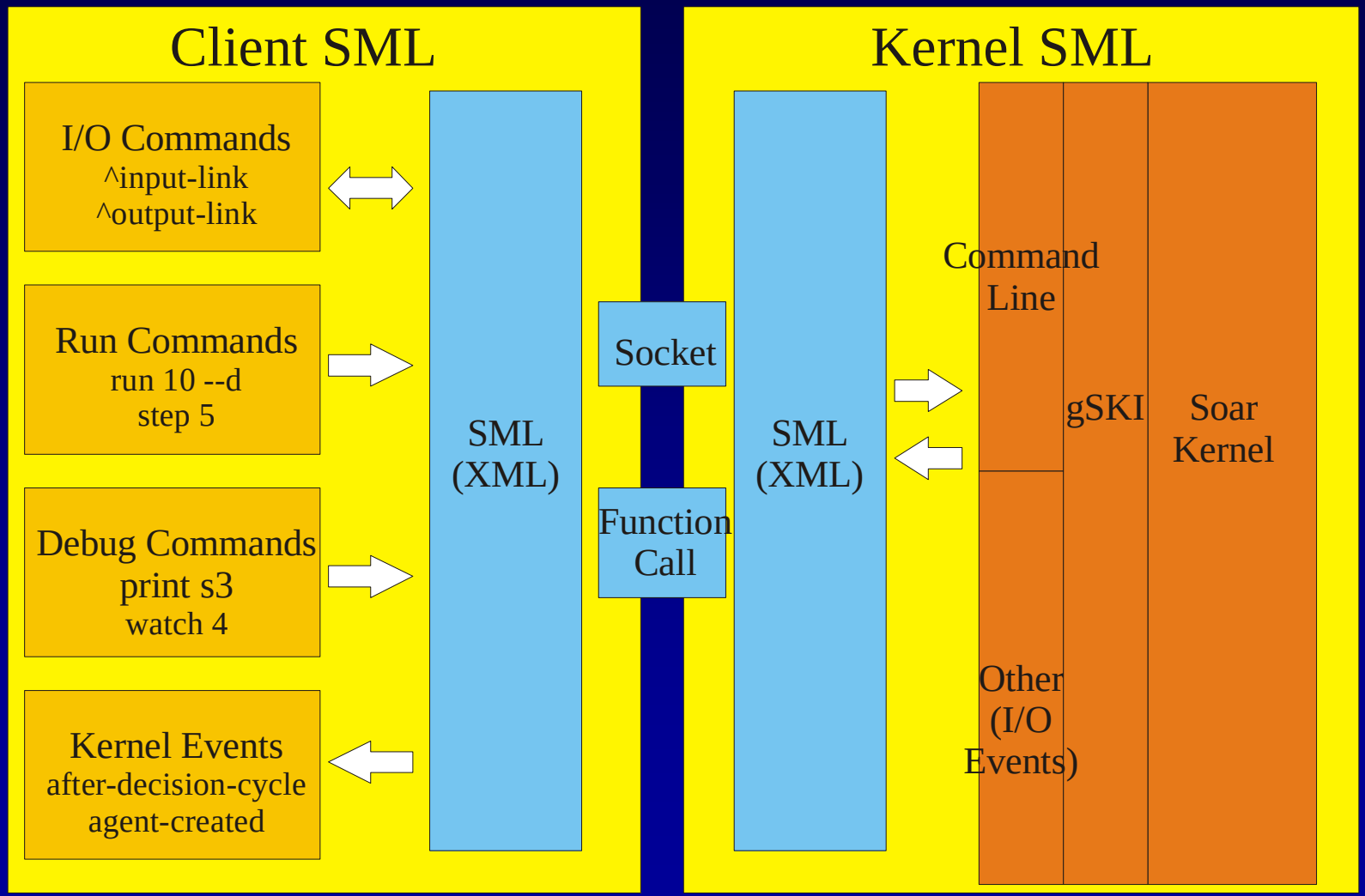
Internally use socket or function call
Client, Kernel and (99%) SML code all unaware

ThreePenny SOFTWARE

Connecting to Soar
SML Style – Soar Markup Language



Connecting to Soar
SML Style – Soar Markup Language



ThreePenny SOFTWARE

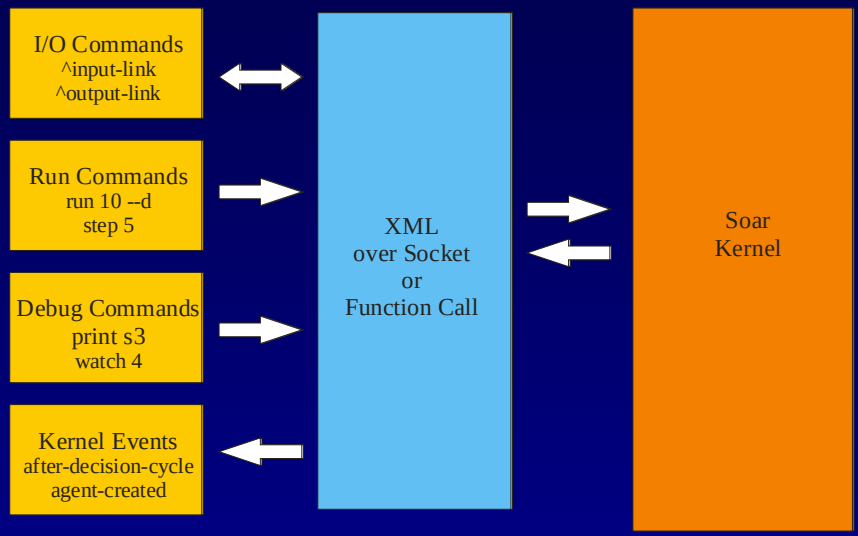
Possibly useful ways to think about SML

- An XML interface into the kernel
  - A data-driven approach rather than function calls
- A remote procedure call protocol for the kernel
  - Although it's not just for remote calls
- A language independent way to access the kernel
- SOAP/Web Service for the kernel
  - Although again not limited to remote access

De-Coupling from the Kernel

- Two central tenets for good software design
  - High cohesion
    - Each class does one thing not several things
  - Loosely coupled
    - Each class knows as little as possible about the classes they interact with
- What is good for classes is good for modules

SML Makes Systems More Loosely Coupled to Soar



8.5/gSKI Interface size (# functions which break client if changed and not recompiled) ~= 1000

SML reduces this to 2 (+ about 20 for ElementXML)

Why does this matter?
Supports a larger community building clients – may not have source available
Allows a single client to support multiple different kernel versions because data-driven

By default builds to 2 libraries with no dependencies
So less coupled to Tcl or other stuff

ThreePenny SOFTWARE

SML Example Packets

“Print chunk-1”

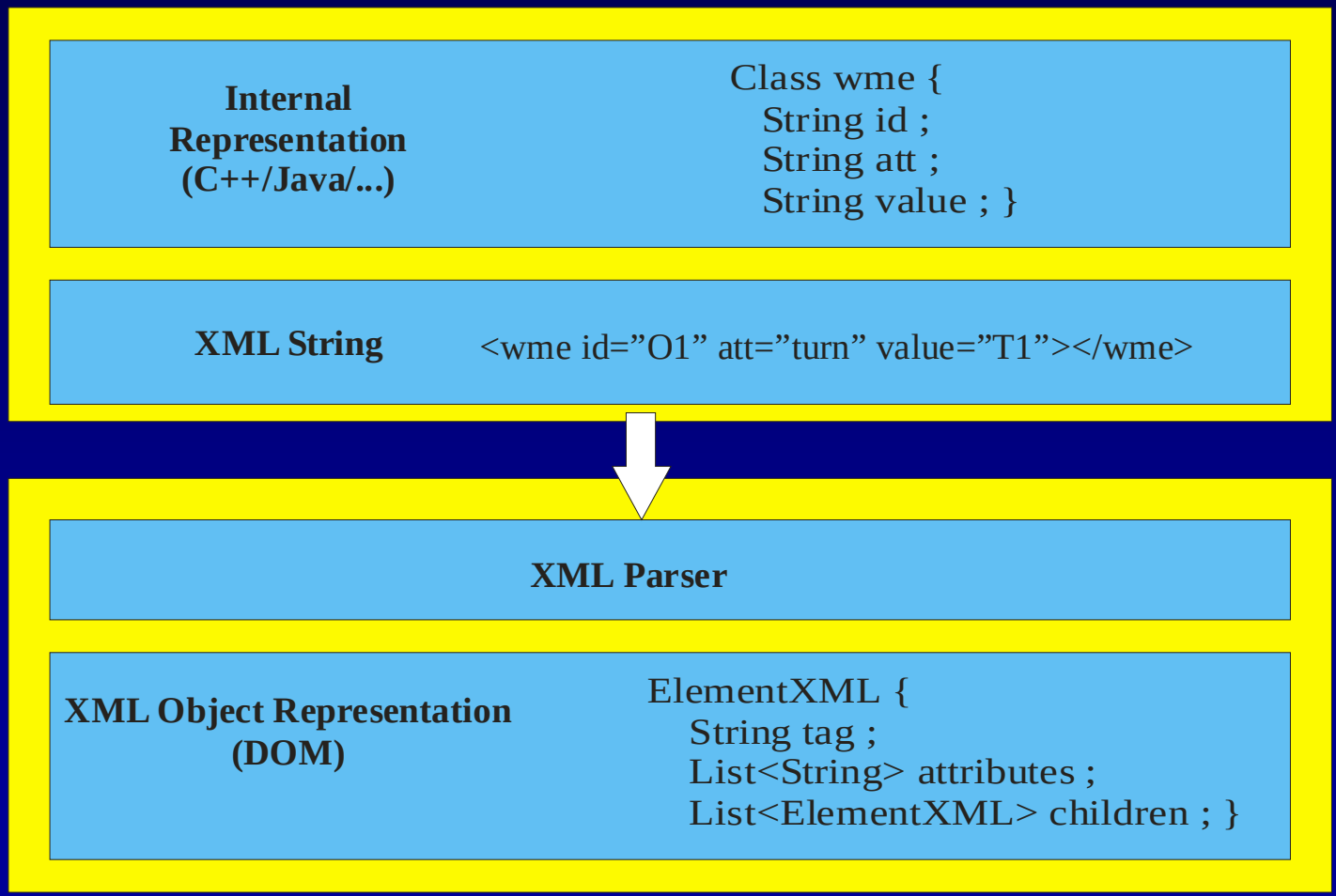
```
<sml smlversion=”1.0” doctype=”call” soarVersion=”8.6.1” id=”657” >
    <command name=”print”>
        <arg param=”agent”>agent-1</arg>
        <arg type=”string” param=”name”>chunk-1</arg>
    </command>
</sml>
```

“Output (O1 ^turn T1) (T1 ^heading 045 ^speed 225)”

```
<sml smlversion=”1.0” doctype=”call” soarVersion=”8.6.1” id=”1763”>
    <command name=”output”>
        <arg param=”agent”>agent-1</arg>
        <wme action=”add” att=”turn” id=”O1” tag=”7” type=”id” value=”T1”></wme>
        <wme action=”add” att=”heading” id=”T1” tag=”6” type=”int” value=”045”></wme>
        <wme action=”add” att=”speed” id=”T1” tag=”12” type=”int” value=”225”></wme>
    </command>
</sml>
```

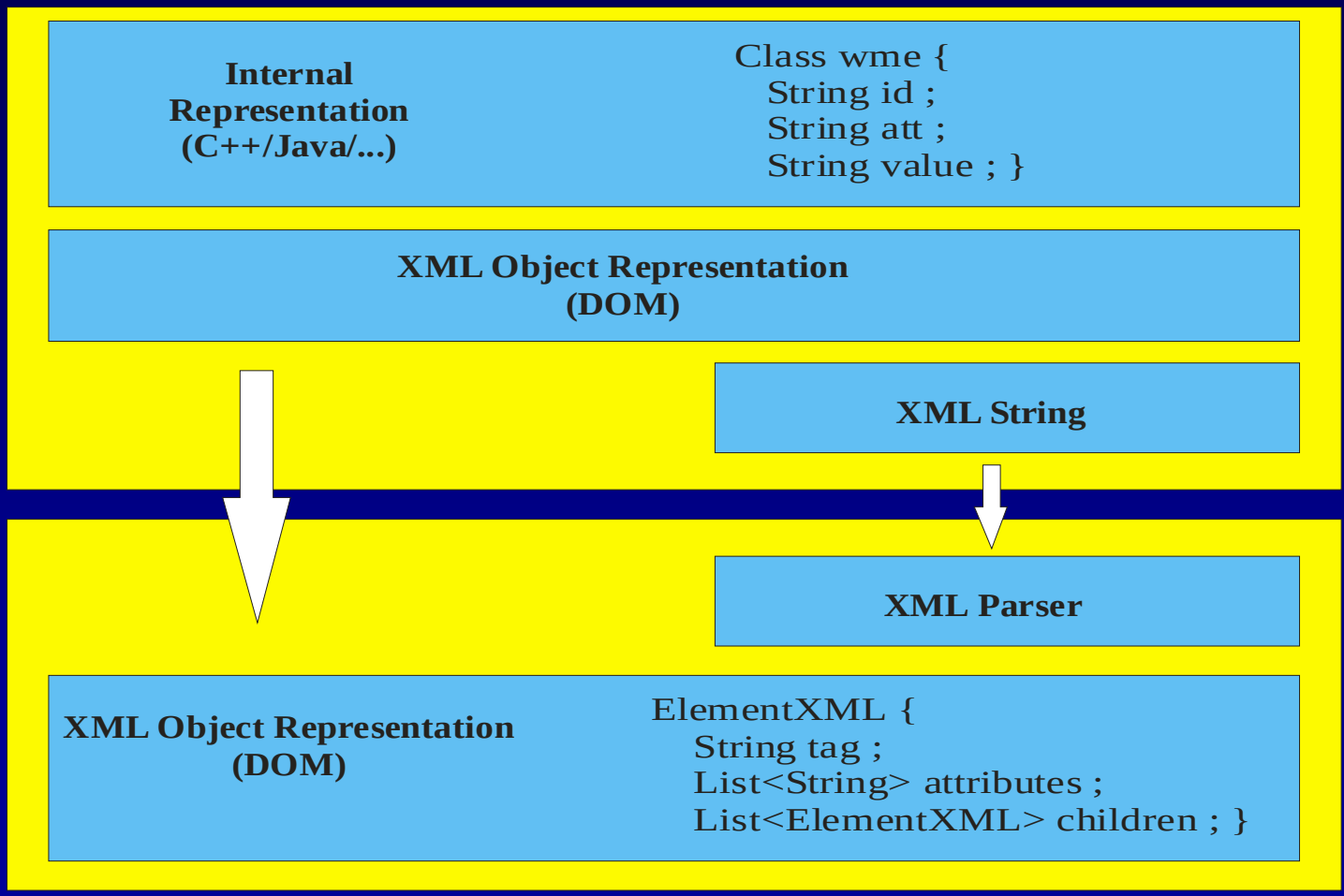
Maximizing XML performance

Standard Approach for passing XML
-- Always requires parsing step



Maximizing XML performance

SML Approach – create DOM representation directly
- only convert to XML string if needed



Speed benefit ~50x if always use socket
Speed benefit ~10x if pass string directly

ThreePenny SOFTWARE

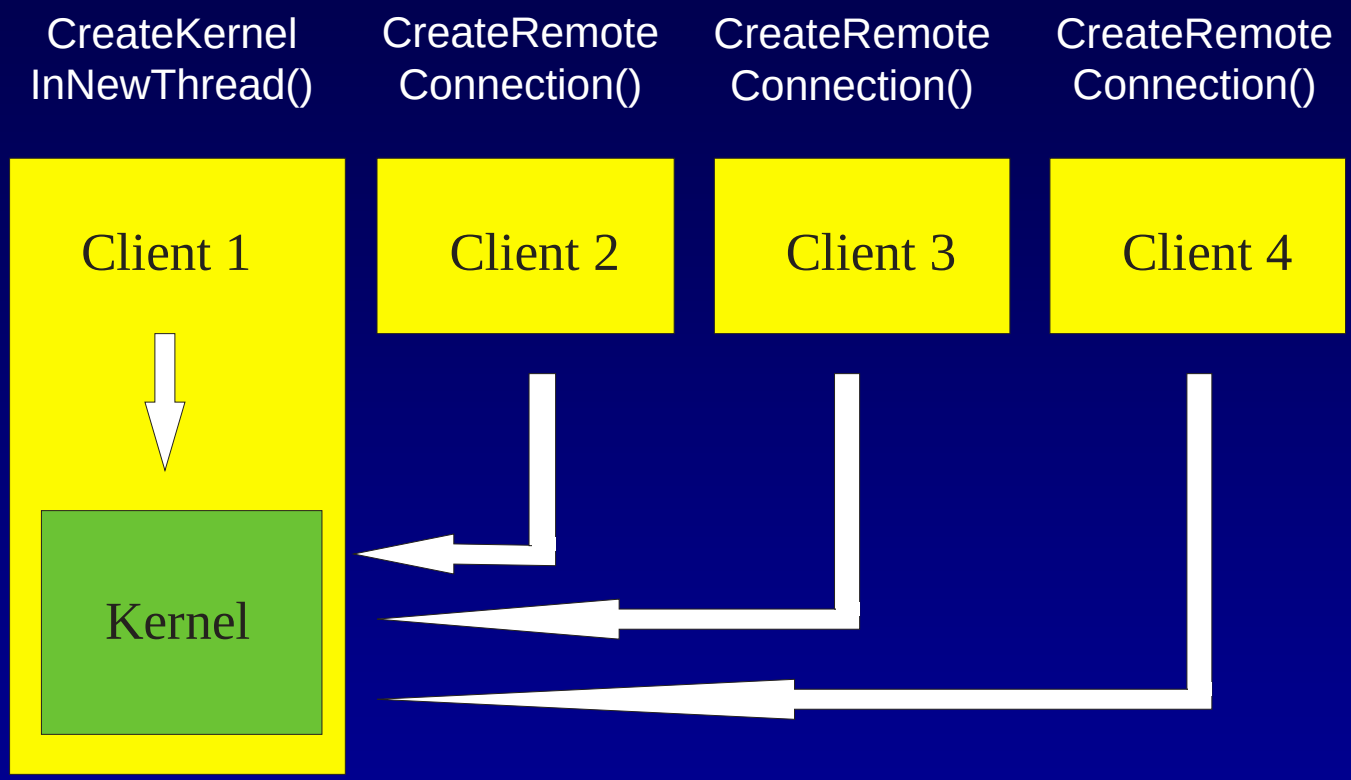
Building an SML Client

- Option A: Just send XML strings to socket
  - Format: 4 byte length + XML string
  - Not recommended, lose embedded speed and need details on XML packets but an option
- Option B: Use client SML functions we provide in
  - C++
  - Java
  - Tcl
  - more…

Creating Kernel and Agents

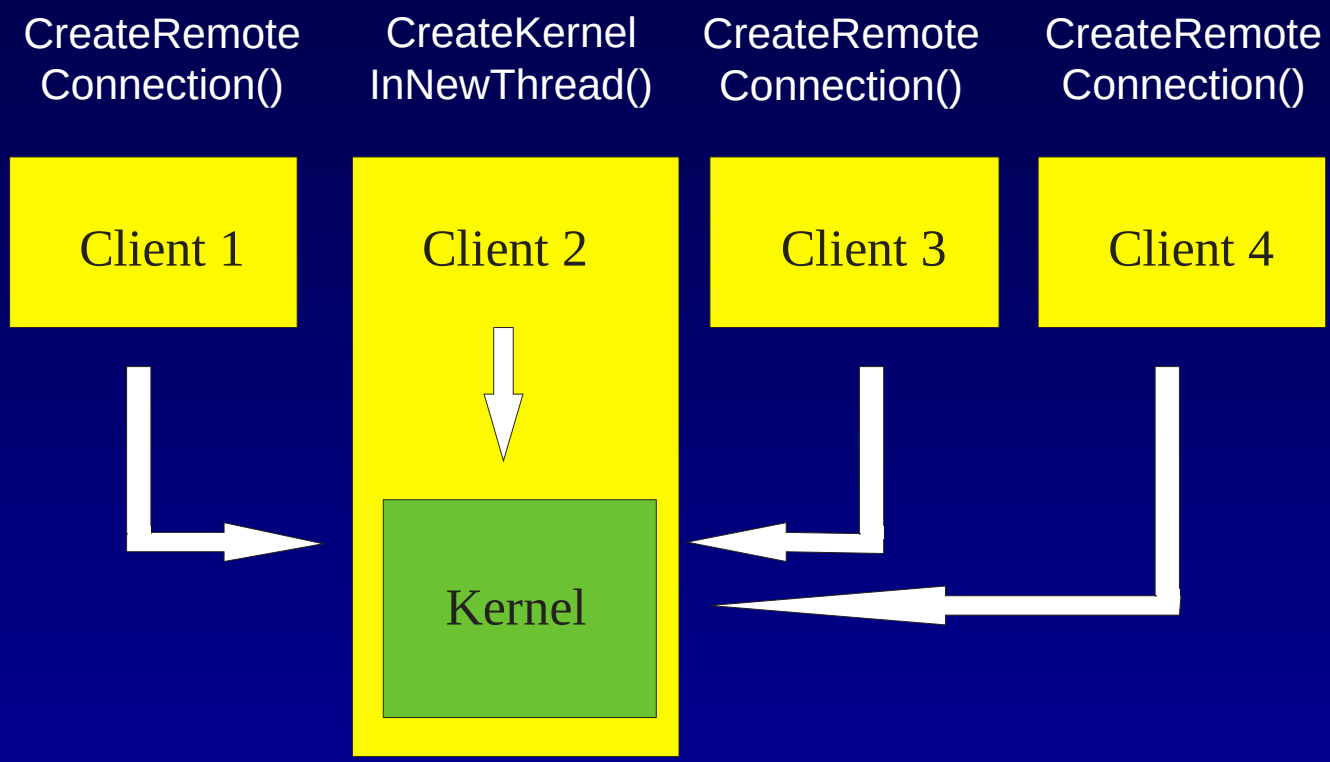
- Local Kernel
  - Kernel* pKernel = CreateKernelInNewThread()
    - Generally recommended
  - Kernel* pKernel = CreateKernelInCurrentThread()
    - Tcl only supports this form (so far)
    - Call CheckForIncomingCommands() periodically
- Remote Kernel
  - Kernel* pKernel = CreateRemoteConnection(ipAddress, port)
    - Pass null for ipAddress => same machine
- Agents
  - CreateAgent()
  - DestroyAgent()
- Shutdown
  - delete pKernel object
  - disconnects
  - cleans up all memory

Local vs Remote Kernels



ThreePenny SOFTWARE

Local vs Remote Kernels



Change one line of code

- Internally different execution paths
  - XML objects + local functions vs XML strings + sockets
- To the client, same interface and capabilities
- No special access for debugger

ThreePenny SOFTWARE

Input

- Add structures to input link representation
  - Identifier* pInputLink = pAgent->GetInputLink()
  - Identifier* pID = pAgent->CreateIdWME(pInputLink, "plane") ;
  - StringElement* pWME1 = pAgent->CreateStringWME(pID, "type", "Boeing747") ;

  Result: (I1 ^input-link I2) (I2 ^plane P1) (P1 ^type Boeing747)

- Send changes to kernel
  - pAgent->Commit() ;
  - Actual wmes added during next input phase

- Update or remove in future cycles
  - pAgent->Update(pWME1, “Cessa”) ;

- NOTE: init-soar works and comes for free.
  - Sends over current input state automatically so you can pick up reasoning

Output

- Check for output commands (since last check)
  - Based on “command” notion on output link
  - E.g. (I1 ^output-link I3) (I3 ^move M1) (M1 ^speed 20)

  - pAgent->GetNumberCommands() ;
  - Identifier* pCommand = pAgent->GetCommand(i) ;
  - string name = pCommand->GetCommandName() ;
  - string speed = pCommand->GetParameter(“Speed”) ;
  - pCommand->AddStatusComplete() ;
    - Same as pAgent->CreateStringWME(pCommand, “status”, “complete”) ;

- But not forced to use this model
  - GetCommand() returns Identifier*. Can access substructure directly.
  - GetOutputLink() and walk output link directly.
  - GetOutputLinkChange() to walk list of WMEs changed

- Call ClearOutputLinkChanges() after reading the output link
  - Allows SML to only report changes to the output link.

Running Soar

- Run
  - Kernel->RunAllAgentsForever()
  - Kernel->RunAllAgents(steps, stepSize)
  - Agent->RunSelfForever()
  - Agent->RunSelf(steps, stepSize)

- Stop
  - Kernel->StopAllAgents()
  - Agent->StopSelf()

Debugging commands

- ExecuteCommandLine(commandLine)
  - ExecuteCommandLine(“watch 3”) ;
  - ExecuteCommandLine(“print o1”) ;
  - ExecuteCommandLine(“excise –all”) ;

- Simple form returns a string
  - “set-library-location”
  - returns: “Current library location: e:\soarmich\soar-library”

- Alternative form returns XML
  - “set-library-location”

```
<sml doctype="response" id="32" smlversion="1.0 soarversion="8.6.1“ack=“237">
    <result>
        <arg param="directory" type="string">e:\soarmich\soar-library</arg>
    </result>
</sml>
```

  - GetArgValue(“directory”) [smlNames::kParamDirectory]

- Why not implement all as methods in the client interface?
  - Just saving work
  - Some clients (lots?) will want to embed command line windows
  - XML all documented on Wiki (http://winter.eecs.umich.edu/soarwiki/Main_Page)

Event Handling

- Register callback handler for an event

  pAgent->RegisterForRunEvent(smlEVENT_AFTER_DECISION_CYCLE, MyRunEventHandler, 0) ;

- Called back when event occurs:

  void MyRunEventHandler(smlRunEventId id, void* pUserData, Agent* pAgent, smlPhase phase) ;

- Supports all gSKI events (plus a few additions)
  - Long list

- Events can be handled over remote connections the same as local ones

RHS Functions

- Register RHS function

```
pKernel->AddRhsFunction("test-rhs", &MyRhsFunctionHandler, 0) ;
```

- Referred to via “exec” in production (just like “tcl”)

```
sp {apply*user*exec
    (state <s> ^operator <o> ^io.output-link <ol>)
    (<o> ^name move ^space <sp>)
    (<sp> ^row <row> ^col <col>)
-->
    (<ol> ^test (exec test-rhs | hello | <row> | world |))}
```

- Calls RHS function handler

```
std::string MyRhsFunctionHandler(smlRhsEventId id, void* pUserData, Agent* pAgent, char
    const* pFunctionName, char const* pArgument) ;
```

- “cmd” allows access to command line as built in command set

```
--> (cmd print <s>)
```

- Handler can be in any language (e.g. register from Java, handler in Java)
- RHS functions can be handled over remote connections the same as local ones
  - If same function registered locally and remote, calls local one

Threads

- CreateKernelInNewThread()
  - Runs kernel in its own thread
  - Means kernel can respond to remote commands (over socket) on its own
  - Otherwise need to call “CheckForIncomingCommands()” periodically

- Also EventThread helps client remain responsive
  - Register for an event (e.g. AFTER_DECISION_CYCLE)
  - Go to sleep => entire system locks up waiting for this client
  - Solution is a thread which receives these events and pushes them through client callback to get response

- Both just make it easier to build a client
  - Neither thread is required – CreateClientInCurrentThread()
  - For maximum performance turn them off and handle these issues yourself

SWIG and Language Support

- SWIG
  - Automatic generation of Java and Tcl implementations
  - Fast to create and maintain
  - Custom code added to handle callbacks
  - Code looks the same in each language

C++

```
sml::Kernel* pKernel = Kernel::CreateRemoteConnection(true, null, Kernel::GetDefaultPort()) ;

if (pKernel->HadError())
{
    cout << pKernel->GetLastErrorDescription() << endl ;
    return false ;
}
```

Java

```
sml.Kernel kernel = Kernel.CreateRemoteConnection(true, null, Kernel.GetDefaultPort()) ;

if (kernel.HadError())
{
    System.out.println(kernel.GetLastErrorDescription());
    return false ;
}
```

Common Environment Control Loop

Asychronous environments

Act as soon as output is sent from an agent

Synchronous environments

Fixed amount of Soar processing before update world

while (!stopped)

- Run(1) (or Run-til-output)
- Collect-output
- Update-state-of-world
- Send-input

Assumes environment triggers run.

Assumes run(1) completes a decision

Assumes user doesn't issue "stop-soar" (needs to pause/stop environment)

Preferred Environment Control Structure

1. Register for “update world” event (e.g. AFTER_ALL_OUTPUT_PHASES)

2. Handler takes form:
   - Collect-output
   - Update-state-of-world
   - Send-input

3. Run controls
   - Kernel::RunAllAgentsForever()
   - Kernel::RunAllAgents(1)

As agents run, fire event as reach end of output phases.

Allows run to come from debugger or environment or other tools.

Allows for arbitrary interruption of run commands (e.g. breakpoints/stop-soar).

Debugger Performance Comparison

Towers of Hanoi	TSI (8.5.2)	Java Debugger (Text)	Java Debugger (Tree)
Watch 1 (Run 100)	1.25 secs	0.73 secs	0.75 secs
Watch 5 (Run 100)	59.68 secs	2.14 secs	1.51 secs 0.58 secs (full filtering)

- Within process faster than 8.5.2 (for the debugger)
- Embedding Soar in environment so remote => local
  - Not measured yet, but huge speed up

ThreePenny SOFTWARE

Documentation

- PDFs included in Release
  - SML Quick Start Guide
  - Moving from SGIO to SML
  - Soar XML Interface Specification
  - Intro to the Soar Debugger
  - How to guide – adding an event to gSKI and SML

- Online on the Wiki
  - http://winter.eecs.umich.edu/soarwiki/Main_Page

- Questions, bugs, building something cool, need help?
  - soar-sml-list@umich.edu

Nuggets

- Supports multiple languages (Java, C++, Tcl currently) while removing Tcl dependency
- Supports uniform interface for I/O (environments) and commands (debuggers)
- Supports embedding kernel within debugger or environment with remote connections between them
- Supports multiple clients (environments, tools, debuggers etc.) connecting to a single kernel
- Supports dynamic connection and disconnection of tools (esp. debuggers) from a running kernel
- Provides a uniform, high-level, data-driven model for the entire Soar interface while achieving high performance
- Moves command line support out of the kernel while providing universal access to it from any client
- Includes a new cleaned up command line interface

- Lots of new capabilities yet in most cases the new interface is substantially faster than 8.5.2

Nuggets

- Building a command line interface in a client is pretty simple
  - Allows embedding console windows into environment (e.g. games)
- Readily supports other solutions
  - Other debuggers (e.g. TSI-8.6)
  - Other editors (edit-production is generic)
- 8.6.1 will be out by end of June
  - Linux and Mac support
  - Soar7 mode
  - Tcl Eaters
  - Java Missionaries and Cannibals
  - New environment event model
  - Kernel XML generation much faster
  - New debugger features (fast tree view, filtering etc.)
  - Lots of other bug fixes
- 8.6.1r5 Used for tutorial at the workshop w/o problems
  - Tcl Eaters with Java Debugger with C++ Kernel

Coal

- Not all commands generate fully structured XML
  - Esp. print, but should do so shortly
  - Still XML but not as rich as we'd like
- Threading models not completely resolved yet
  - Esp. Tcl
- Lots of new code
  - There will be some bugs lurking
  - Not all combinations fully tested yet
  - Interfaces will probably change a bit as everything settles