



December 3-6, 2007, Santa Clara Marriott, Santa Clara, CA

Direct Access CIM Perspectives

Ed Boden

bodeneb@us.ibm.com

IBM



Topics

- The concept
- Why useful
- Key details
- What about ... ?
- Direct access CIM APIs
- Architectural implications & issues

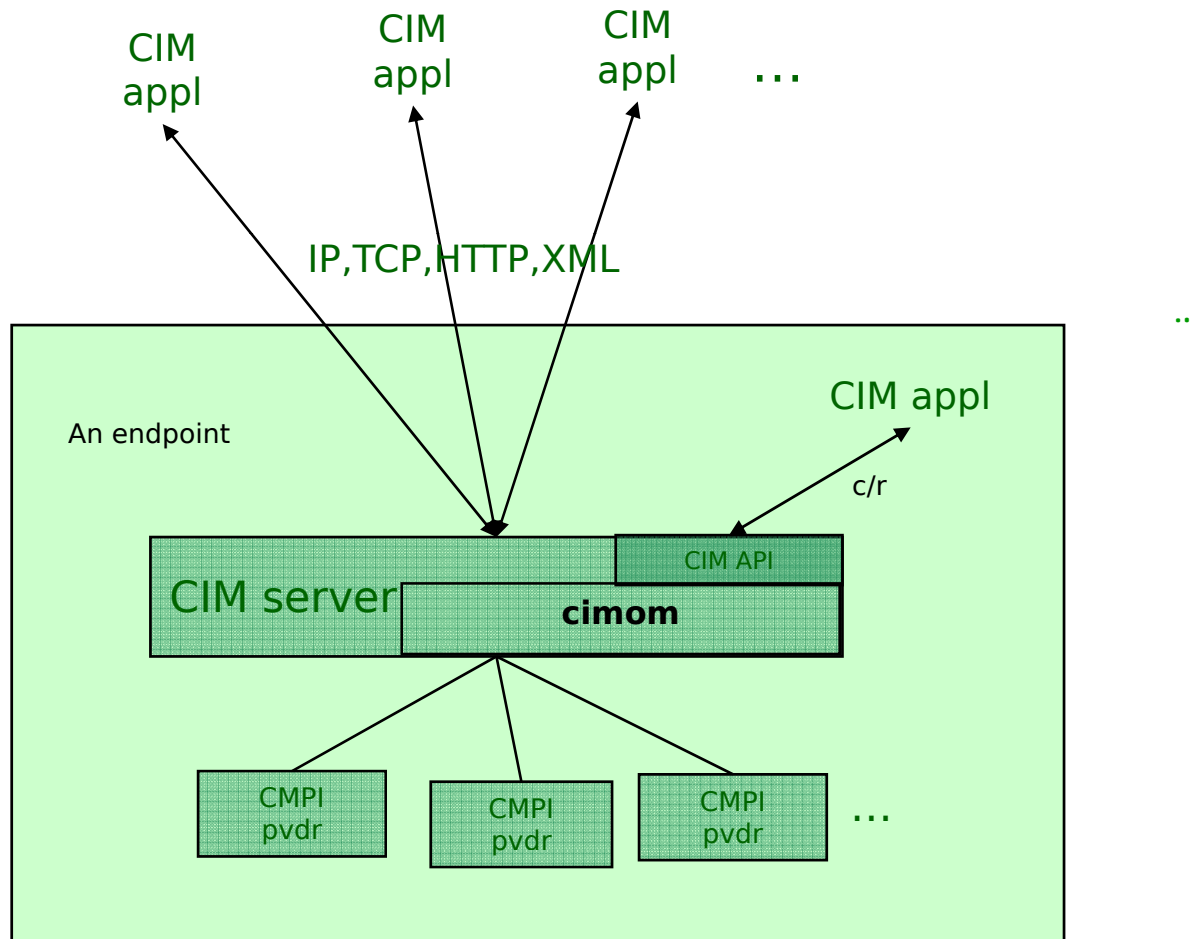


The concept

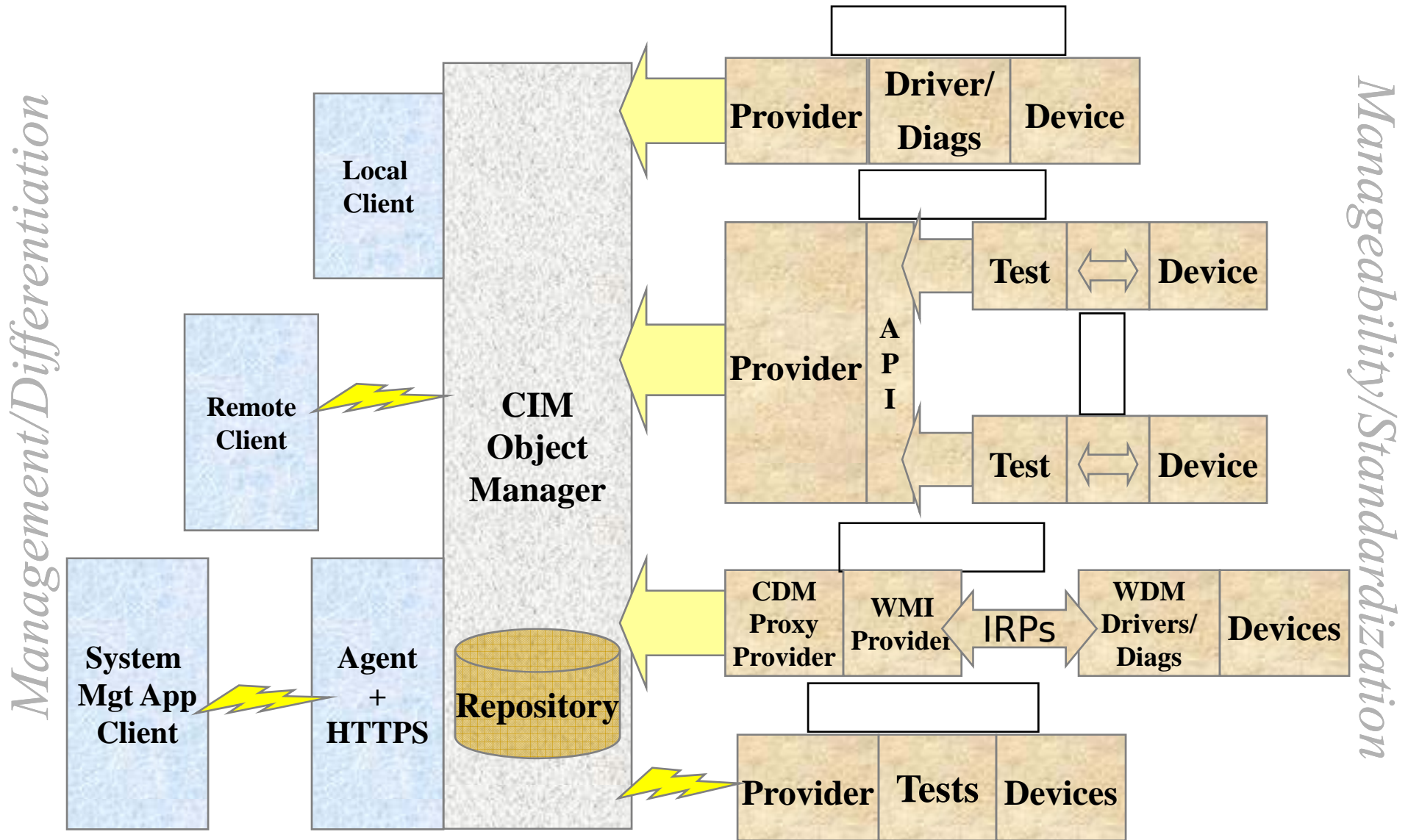
Direct access to local CIM model information

- Small foot print & very fast, low overhead access to CIM providers
- Embeddable CIMOM, linkable to application: no TCP/IP, no HTTP, no XML.
- Providers are implemented in the open standard CMPI
- Support concurrent access to existing providers across threads, processes
- Multiple language bindings
- Implementation focus on CIM instances and related operations
- Support CIM indications
- Optional in-memory repository

The concept



The concept: idealized architecture for CDM requirements



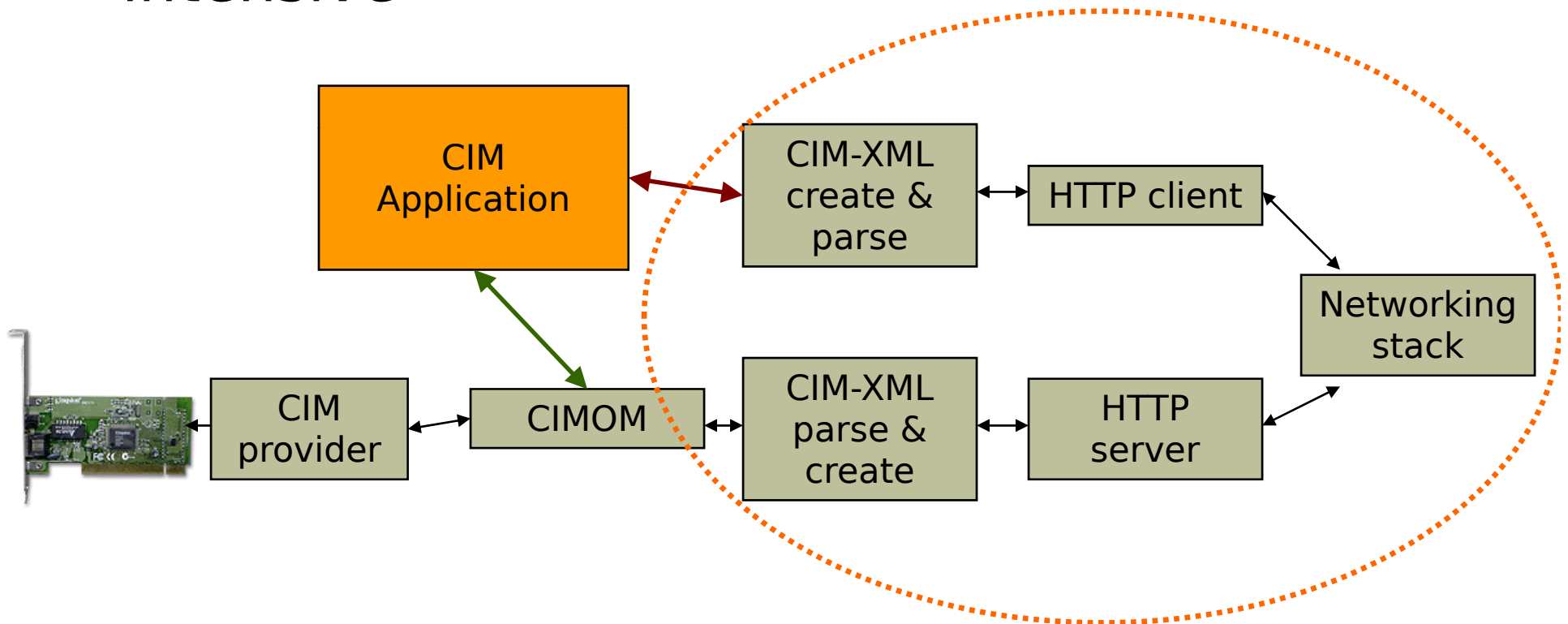


Why useful

- Improved Performance
- Alternate access protocols
- Local java access
- Server Management CLP
- Hierarchical models
- Embedded CIM
- Namespace providers
- Alternate to proxy providers

Key benefit of direct access CIM

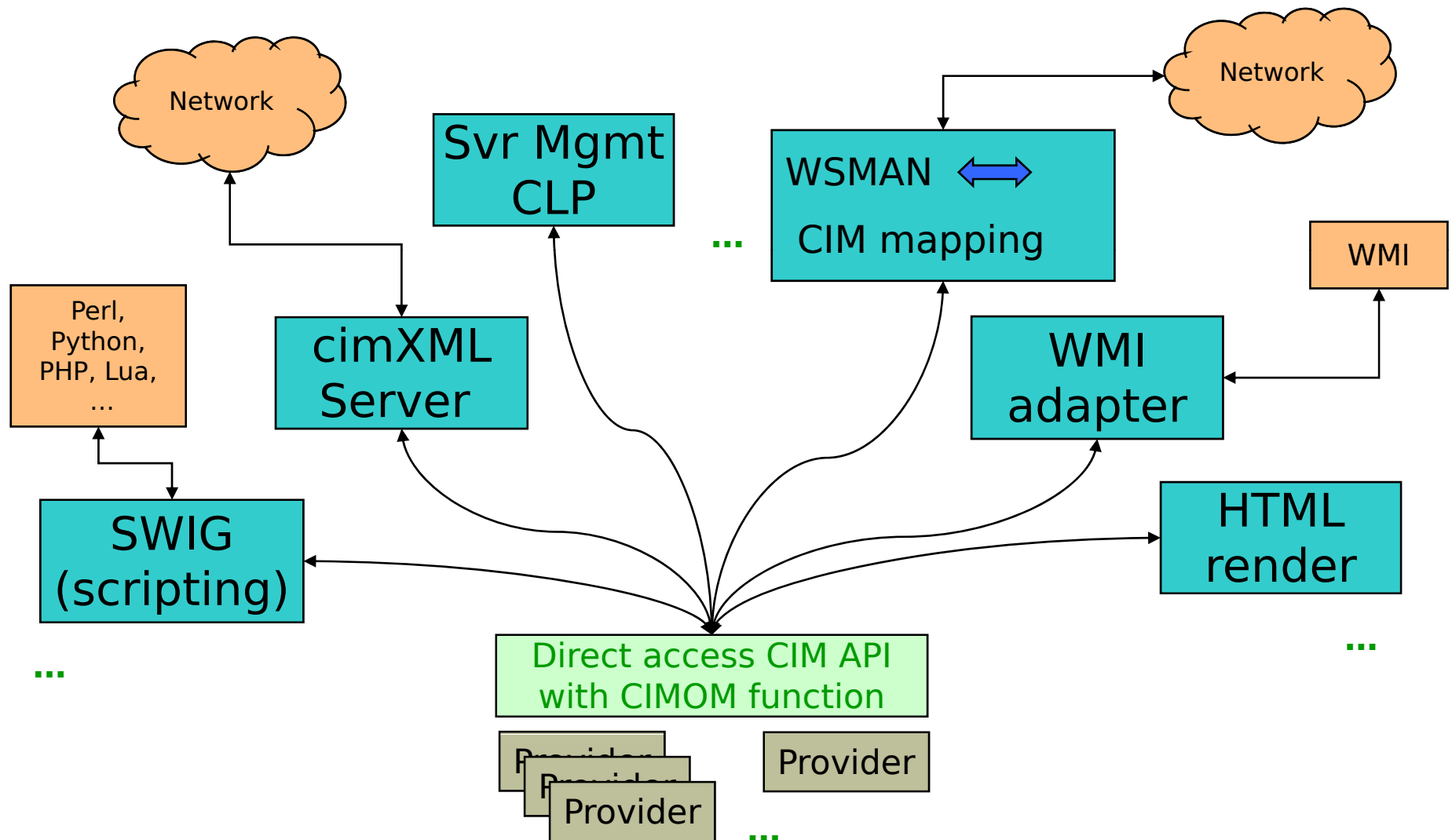
A directly callable interface is less resource intensive



How much?

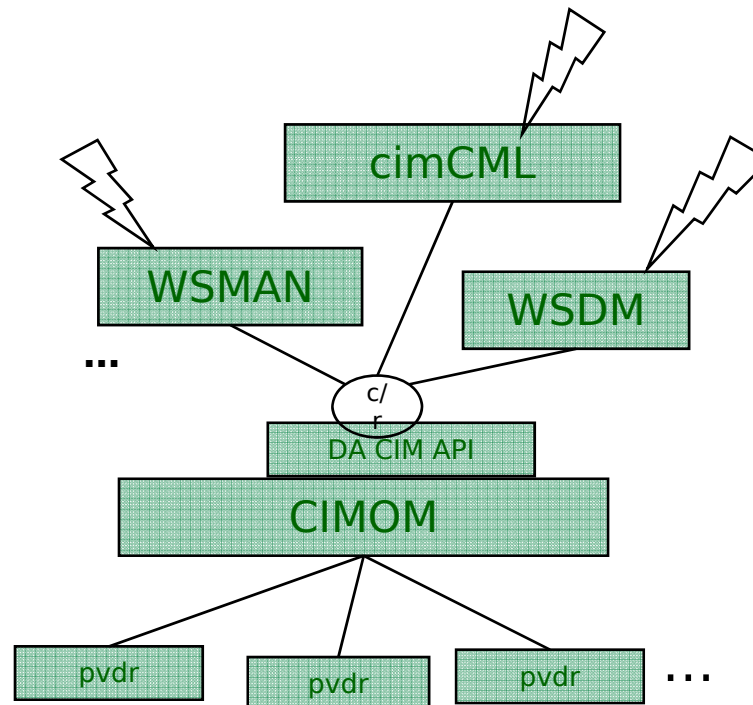
- Direct access CIM is faster
 - than corresponding operations
 - for BAU CIM client, using cimXML
- Basic tests on Beta DACIM implementation
- Beta DCAIM implementation had no performance tuning
- Compared with OpenPegasus 2.7 pre-SR
- Enumerate Instance 100; -68.5%
- Enumerate Instance 1; -37%
- Associator names 1; +4.7%

Examples of Direct access CIM API callers



DACIM use (a)

- Multiple protocols
 - cimXML
 - Web Services Distributed Management
 - Web Services Management
 - Proprietary



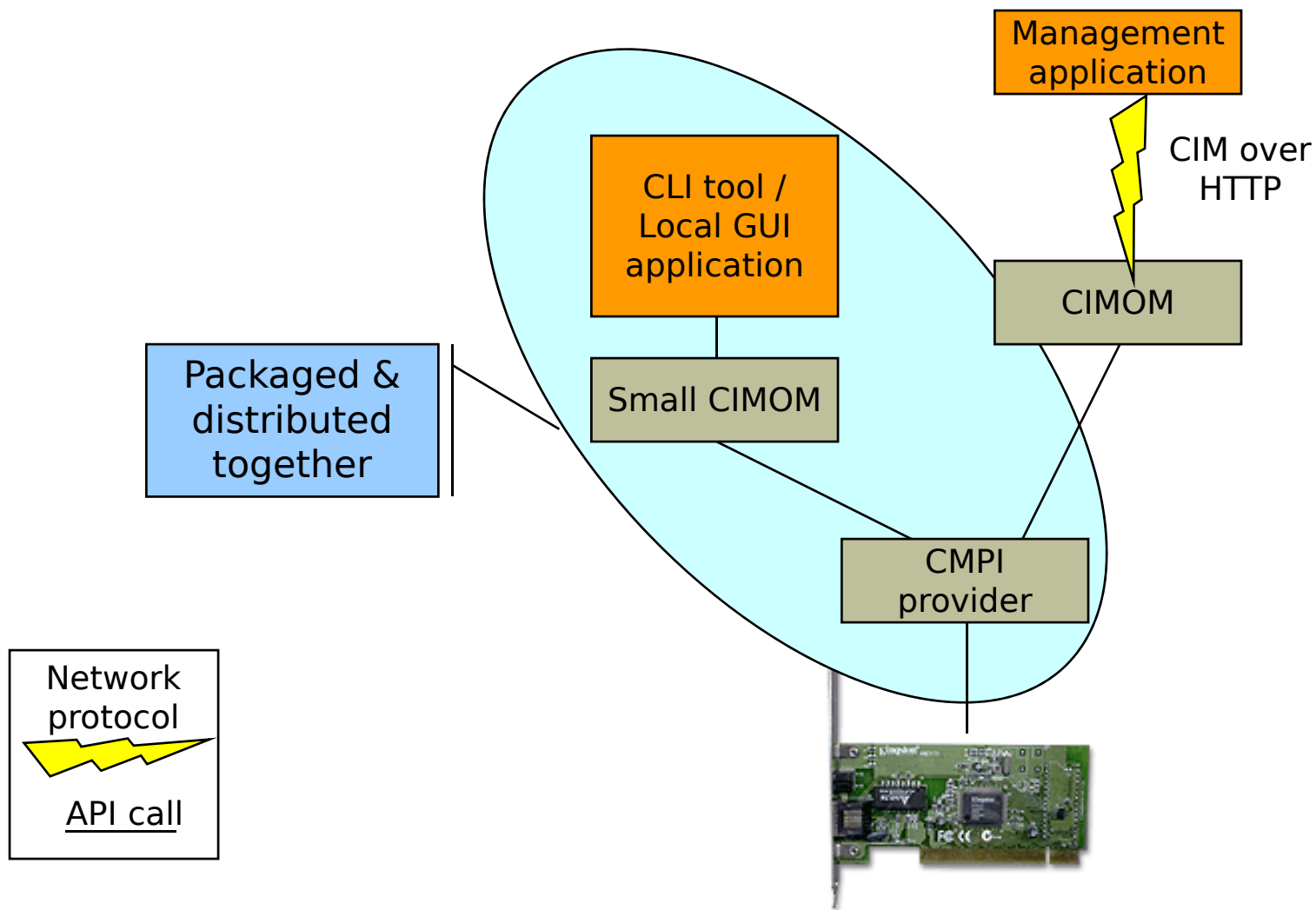


DACIM use (b)

- Local CIM applications
- There is a need for local management applications
 - CLIs, local GUI applications
- These applications sometimes neglect leverage CIM
 - Can't rely on CIMOM and/or necessary CIM providers being installed
 - Customer expectation is that they are self contained, so prereqs not allowed
- Make it simple to create applications using CIM providers as plugins
- SMASH & local java access

Use CIM providers for hardware abstraction

Best





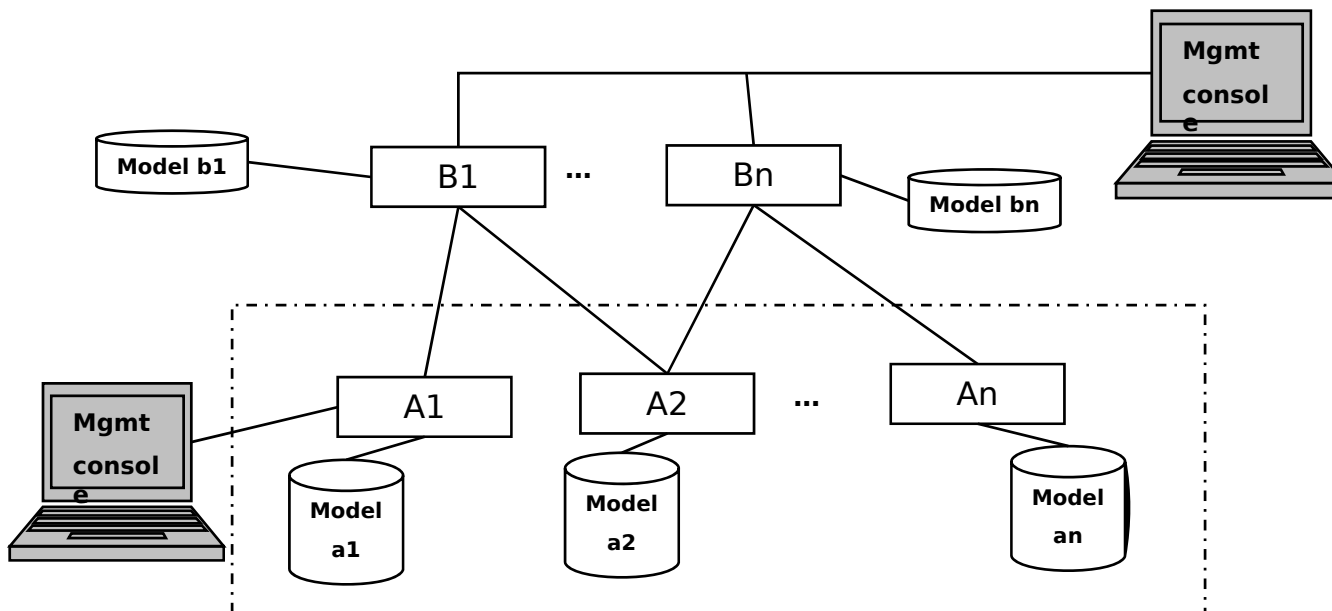
DACIM use: embedded¹

- So-called 'stand-alone' build
- Footprint
- Remove non-operation methods
- Need indications via callback?
- Registrations built-in or dynamic?
 - Third-party providers
 - Provider type(s)
- Repository in-memory or in-cache?
- Shared lib
 - 1 or n
 - Static or dynamic linking

¹ In the sense of local use of CIM, rather the remote use, with CIMOM function as 'part of my application'.

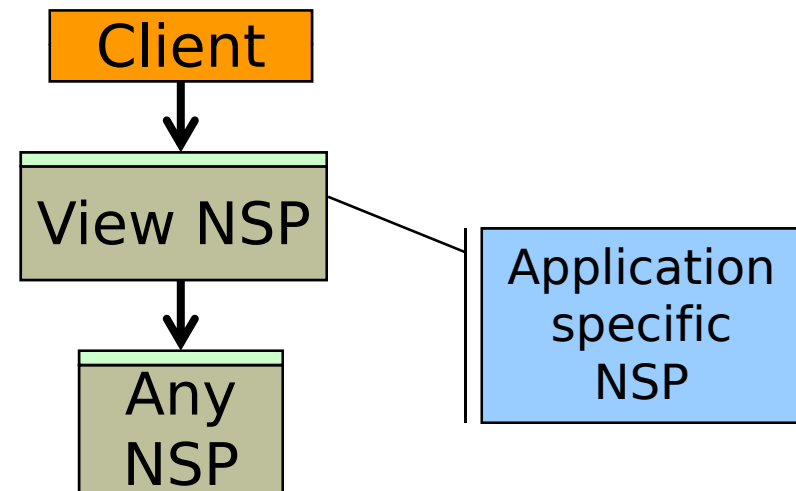
DACIM use (d)

- Hierarchical CIM models (“orchestration of management”)
- At intermediate layers, the CIM ‘application’ is a CIM provider
- Natural boundary for CIM
 - Proxy provider and remote providers
 - Protocol choices
- Seems particularly useful in telecommunications



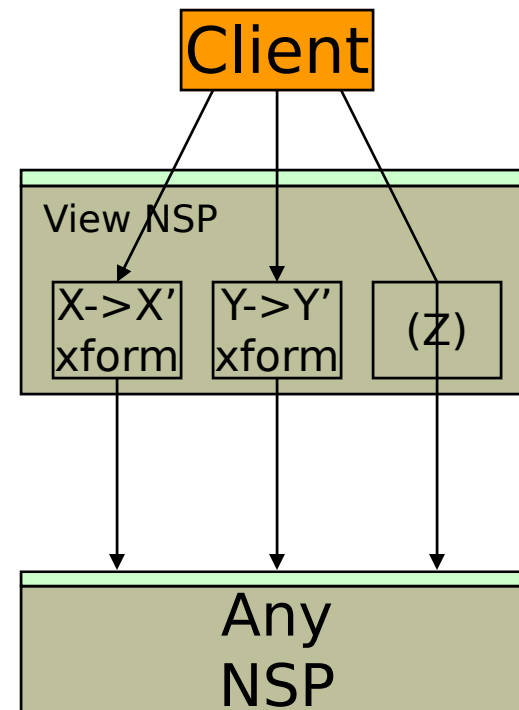
DACIM use (e)

- Namespace providers
- Similar in concept to a database view
- Modifies the schema and/or instances surfaced by underlying NSP

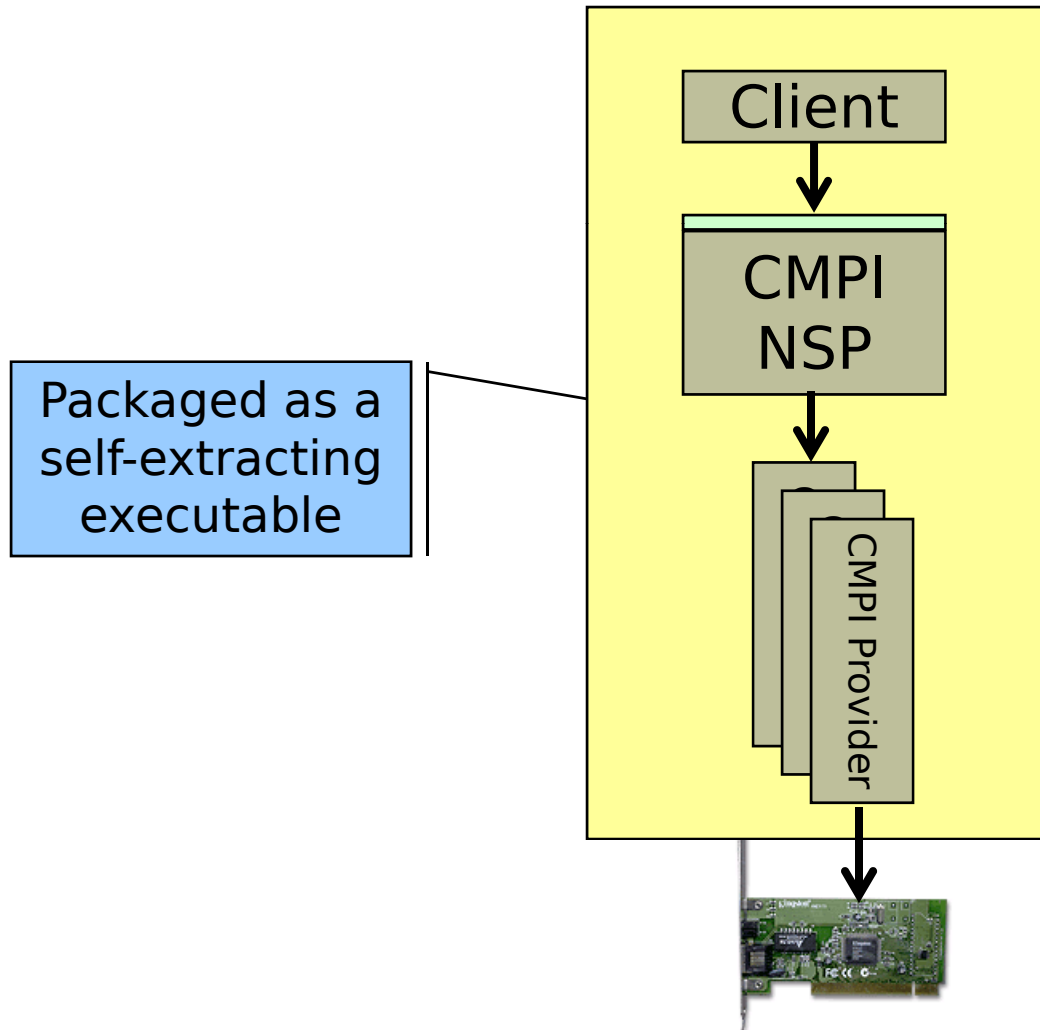


How layering solves this problem

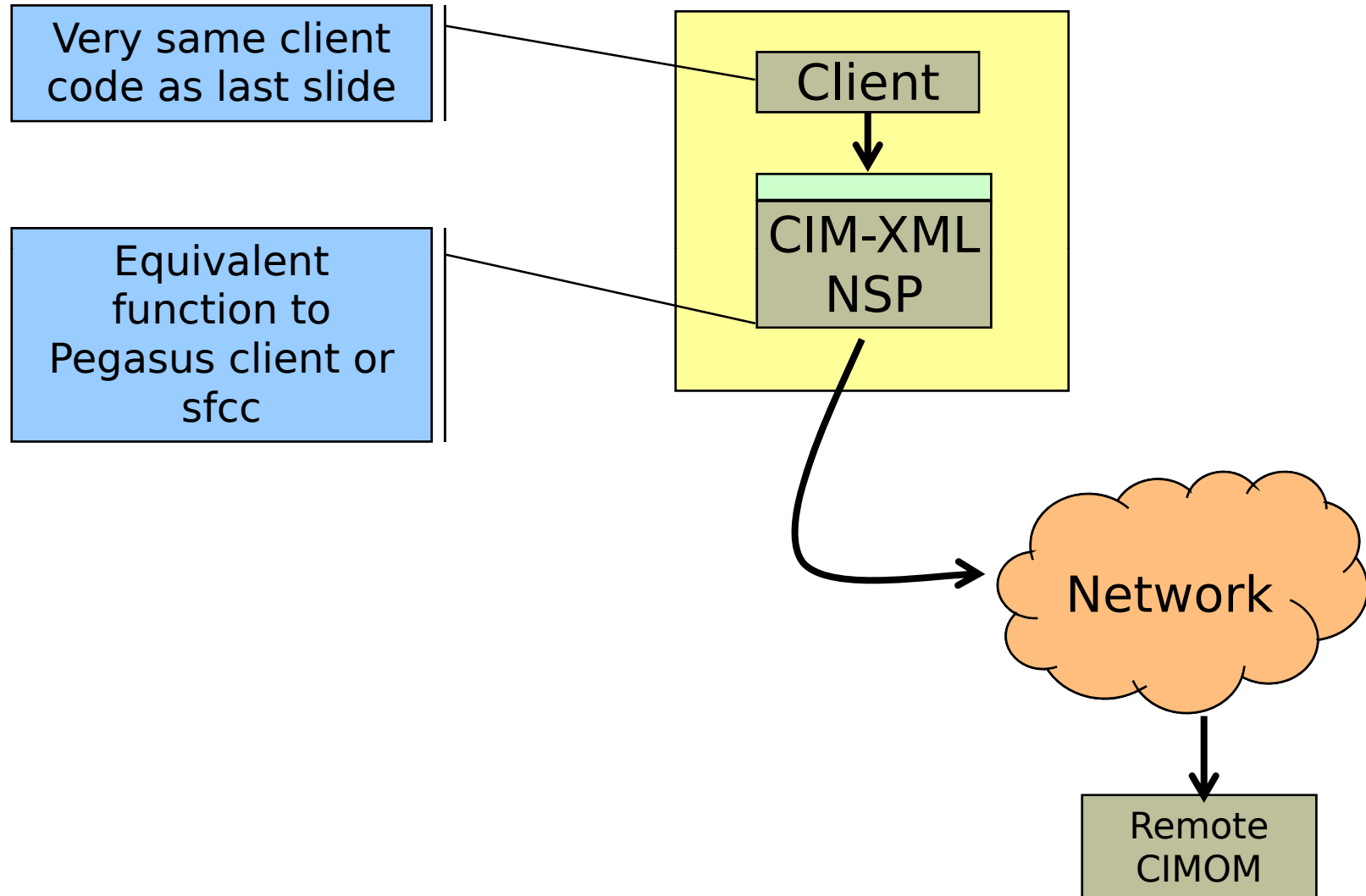
- A View NSP is layered on top of the underlying NSP and houses the transform code
- Client calls for X' and Y' are satisfied by calling the underlying NSP and transforming X into X' and Y into Y'
- Client calls for Z are passed untouched to the underlying NSP



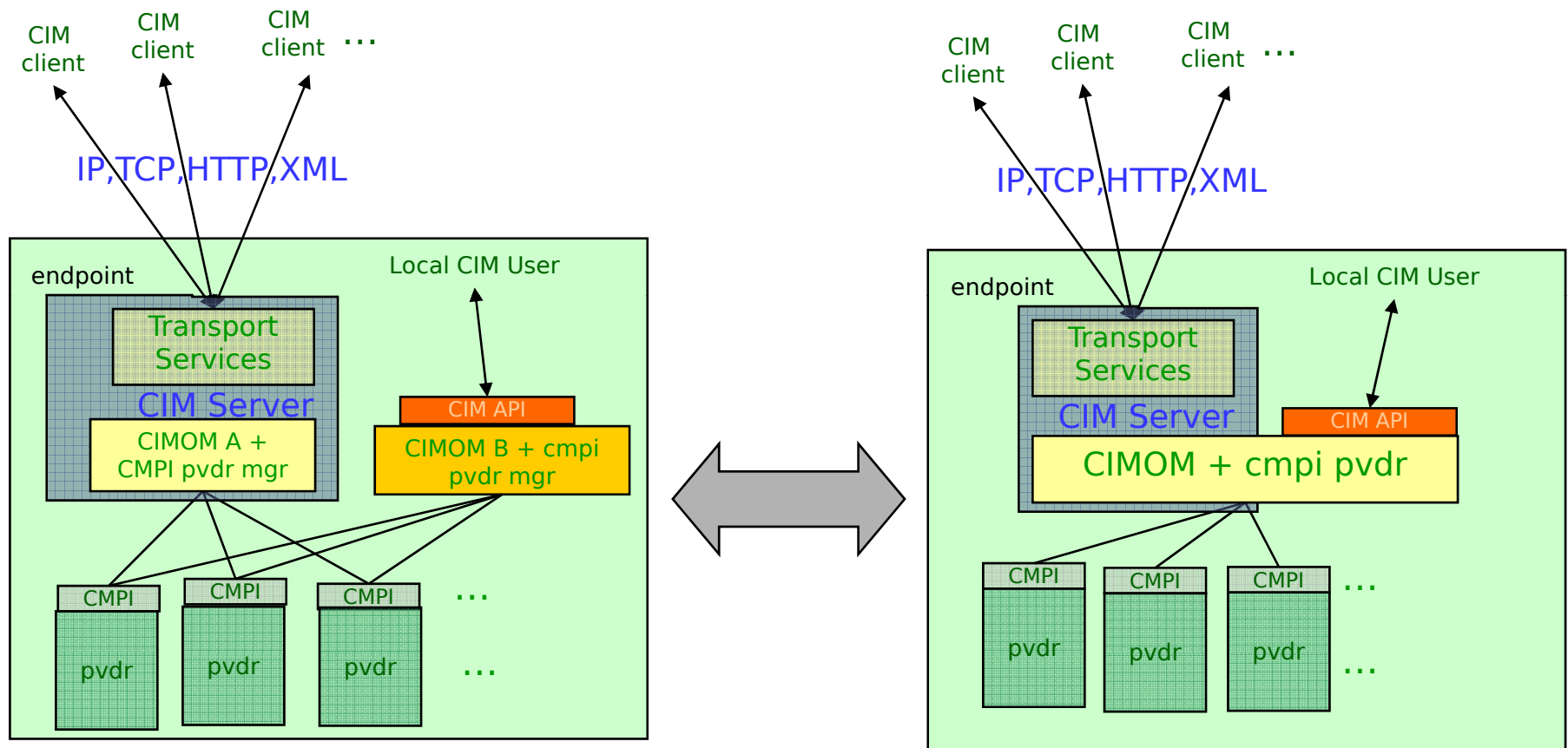
Self contained "tool" to access hardware



Local app can be turned into a remote one by swapping NSPs



DACIM: a range of implementation & rollout possibilities



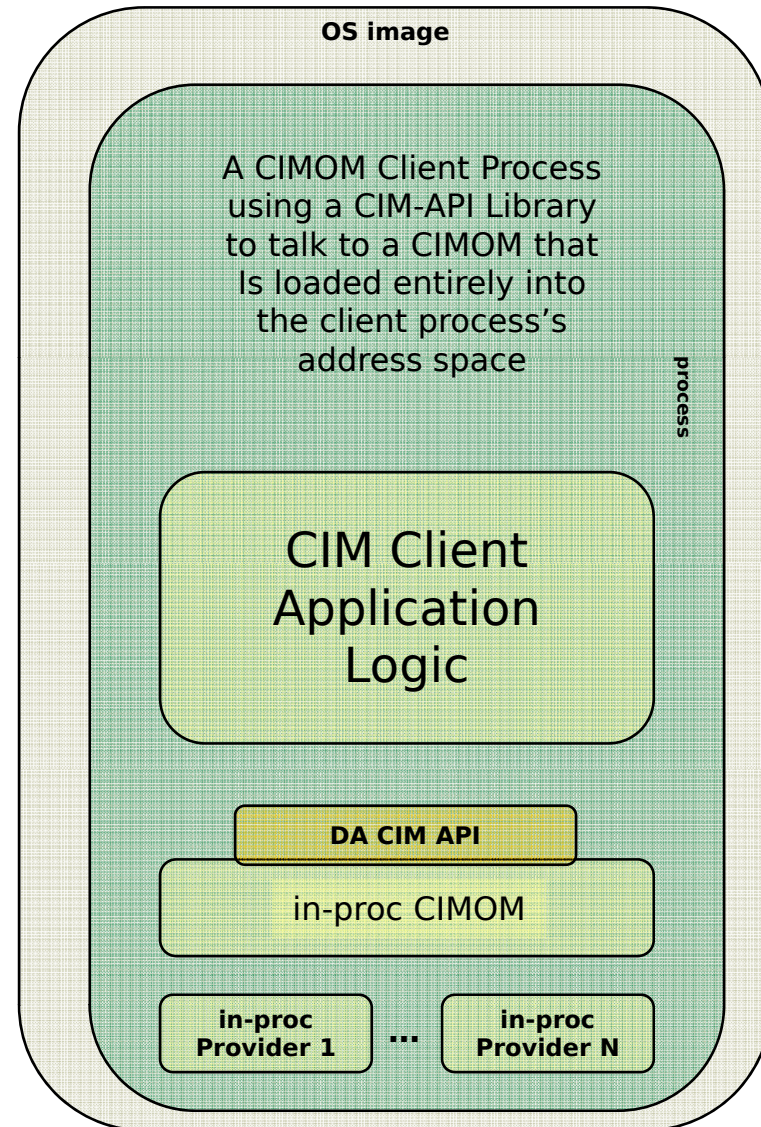
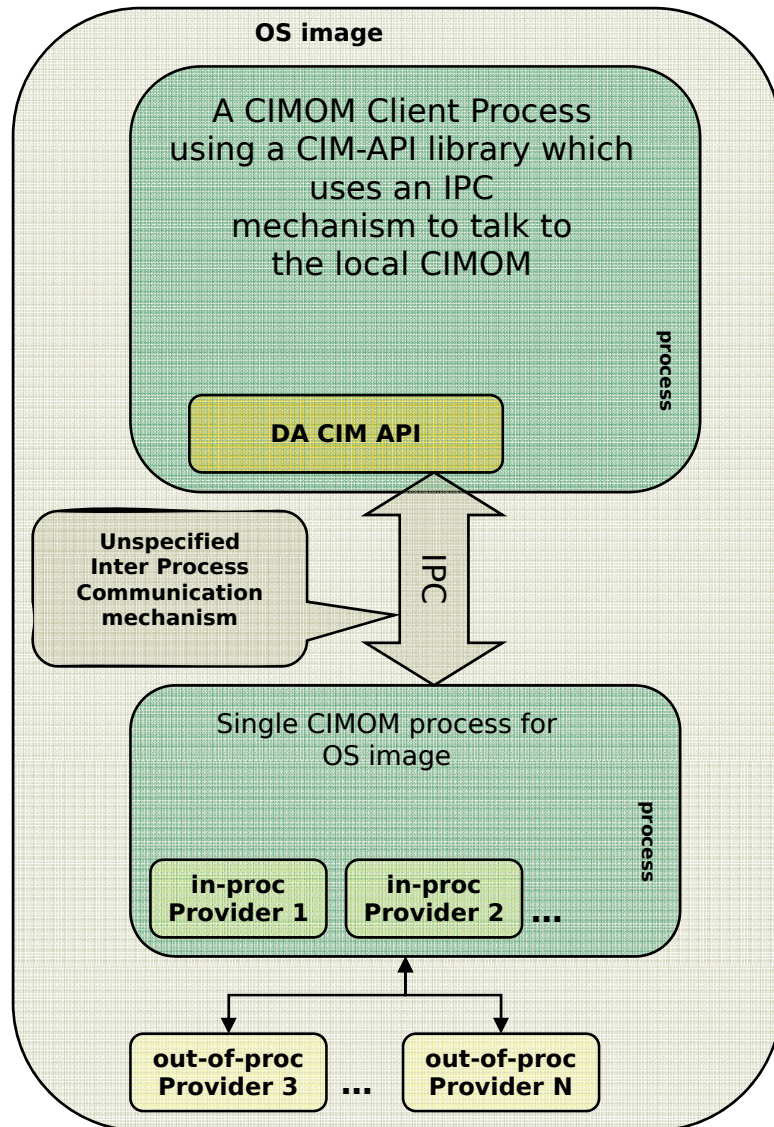
Full separation of design, implementation (no common code) & runtime binaries.

Full unity in design, implementation (100% common code) & runtime binaries.

Key design points

1. Full runtime sharing with a CIM server
 - Repository
 - Registrations
 - Indications
 - CLI
 - Libraries
2. Partial runtime sharing with a CIM server
 - Repository
 - Registrations
3. Standalone (embedded)
 - CIM client API local or remote transparency
 - Authorization
 - Multiple provider types
 - Indications
 - Other; operations, ...

The two basic DACIM process models





DACIM impl: client

(openpegasus)

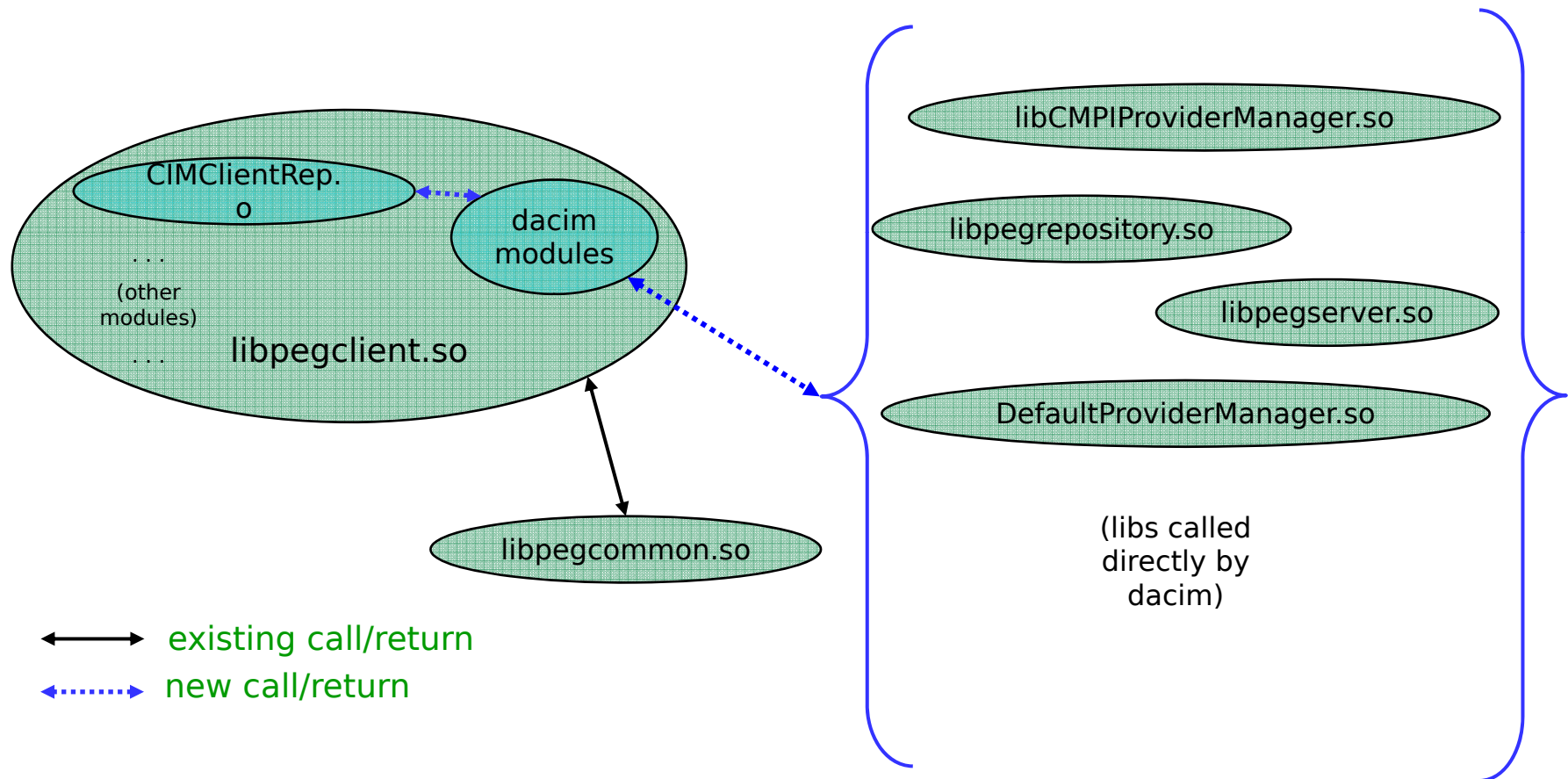
- Beta implementation used C++ CIM client API without change
- Other APIs exist & could be readily added
- Logically, client-side protocol stack is replaced with a call
- Key changes within CIMClientRep.*
 - ctor, dtor
 - _connect()
 - _disconnect()
 - connectLocal()
 - _doRequest()
 - (other)



DACIM impl: server

- CIMDirectAccessRep (singleton) instance is bridge
- CIMClientRep: : _doRequest() calls CIMDirectAccessRep: : dorequest()
- ... dorequest() switches on CIMRequestMessage ->getTYpe()
- ... calling private methods based on CIMOperationRequestDispatcher methods
- ... etc. to Control or regular providers.
- No use of MessageQueue's
- Sequential provider processing (no threading)

Direct Access CIM implementation build structure*



* not to scale; openpegasus 2.7



Beta implementation

(openepgasus)

- Assumptions; full runtime sharing with a CIM server, no local & remote transparency, authorization not applicable, single provider types, no indications, all extrinsic operation
- ~6 kloc

```
src/Pegasus/Ci i ent/CI MCI i entRep. h // daci m cal l s
src/Pegasus/Ci i ent/CI MCI i entRep. cpp // daci m cal l s
src/Pegasus/Ci i ent/CI MDi rectAccessRep. h // new
src/Pegasus/Ci i ent/CI MDi rectAccessRep. cpp // new
src/Pegasus/Ci i ent/CI MDi rectAccess. h // new
src/Pegasus/Ci i ent/CI MDi rectAccess. cpp // new
src/Pegasus/Provi derManagerServi ce/Provi derManagerServi ce. h // fri end dcl
src/Pegasus/Provi derManagerServi ce/Provi derManagerServi ce. cpp // bt & rt
// daci m ck to bypass responseChunkCal l back()
src/Pegasus/Server/CI MOperati onRequestDi spatcher. h // fri end dcl
src/Pegasus/Common/Message. h // 2 new msg types
src/Pegasus/Common/Message. cpp // new msg types & global daci m swi tch
src/Pegasus/Common/PegasusVersi on. h // daci m=
src/Pegasus/Reposi tory/CI MReposi tory. cpp // bui ld both caches as 0 bytes
src/Pegasus/Provi derManager2/Operati onResponseHandl er. cpp
// i n orh:: send(), rt check bypass i sAsync() l ogi c
```



Beta implementation

- Build

- pegasus/mak/config.mak // setup daci m build
- pegasus/readme.directaccessCIM // new
- src/Pegasus/Makefile // build of DirectAccessCIM/ & tests
- src/Pegasus/Client/Makefile2 // daci m libpegclient build with dependencies
- pegasus/Makefile // pegclient 2x, usage fix, convenience targets
- src/Pegasus/Client/Makefile2 // new

- Test

- src/Pegasus/Client/tests/DirectAccessCIM // new
- src/Pegasus/Client/tests/DirectAccessCIM/Makefile // new
- src/Pegasus/Client/tests/DirectAccessCIM/daci mClient.cpp // new
- src/Pegasus/Client/DirectAccessCIM/tests/readme.txt // new
- src/Pegasus/Client/DirectAccessCIM/tests/test.mof // new
- src/Pegasus/Client/tests/Client/Client.cpp // explicit typing of Boolean
- src/Pegasus/Client/tests/Client/Makefile // -D for a trace switch
- src/Clients/TestClient/TestClient.cpp // various fn & convenience
- src/Clients/CLITestClients/CLI/CLI.cpp // pgm=cimcli
- src/Clients/CLITestClients/CLIClientLib/CLIClientLib.h // add alias op names
- src/Clients/CLITestClients/CLIClientLib/CLIClientLib.cpp // fixes



DACIM APIs

- DACIM vs DACIM APIs
- DACIM interface

```
Message *CIMDirectAccessRep::dorequest(  
    AutoPtr<CIMRequestMessage>& request );
```

- DACIM APIs
 - **C** – sfcc, as alternative back-end
 - **C++** -- various exist; OpenWBEM, CIMPLE (internal), OpenPegasus, internal
 - **Java** -- can access any of the C or C++ API's via JCA or JNI.
 - Python? (via swig, a CIM API, or direct?)
- Evolutions
 - Cursor-based operations (WIP CR00386)
 - JSR48 in C++
 - CIMv3
 - ...



What about ...?

- **Indications**
 - If required, exploiting local caller
 - Opportunity to simplify subscription
- **Registration**
 - Range; use Server to statically linked
 - Middle is harder -- CLI
- **Security**
 - Range; closed interface to process to parms
 - Implications for local out-of-process providers
- **Can support multiple concurrent provider types**
 - Relatively simple
- **Alternate schema**



DACIM Indication API

(openpegasus)

- **CIMClient.h**

```
• #ifndef PEGASUS_USE_EXPERIMENTAL_INTERFACES
  #ifndef PEGASUS_USE_DIRECTACCESS_FOR_LOCAL
    //
    //----- DACIM indication API -----//
    //
    void addSubscription( cimSubscription& );
    void removeSubscription( const cimSubscription& );
    void doNotUseDirectAccessForLocal ();
    bool isDirectAccessForLocal ();

typedef void (*indicationListener)( const CIMInstance& );

struct cimSubscription {
    String
        subscriptionName,          /*any name, should be unique. ignored.
        indicationSourceNameSpace, /*NS where indi originates
        filterQueryLanguage,      /* language of filtering expression
        filterQuery;              /* filtering expression
    indicationListener
        indicationCallback;        /*routine to call when an indi happens
    const void *ownerhandle;      /*pointer to whatever the addSubscription()
                                    /* caller wants. It is completely ignored.

    CIMDateTime
        subscriptionAddTime;      /* start is when subscrip is added
    cimSubscription();            /* will set all to default values.
    private:
        pthread_t t_;
};
#endif
#endif
```



Further Architectural implications

- CIMOM as kernel
- Boundary
- Modularity
- Threading

Summary

- Direct access CIM capability gives marked **performance improvement** for many CIM applications
- Direct access CIM is an **architectural enabler** for many use cases & creates opportunities for innovation
- Direct access CIM can simplify and improve the **flexibility & modularity of CIM infrastructure** implementations