Distributed computations with GAP

Steve Linton
Centre of Interdisciplinary Research in Computational Algebra
University of St Andrews

International Workshop “Parallel Programming in GAP’
St Andrews, 18-24 August 2013

What is “distributed” ?

Running several (local or remote) independent copies of computer algebra system(s) to solve problems.

For example:

- GAP and another GAP installation elsewhere
- Several copies of GAP to work in parallel
- GAP and another computer algebra system(s)

Mixing local and remote

- Some software doesn't work on Windows
- Some requires large (and perhaps changing) databases
- Some is still under development and you want to use the latest version
- Some you didn't realise you need before you left home
- Some may only be released as an online service

- Commonly used: ssh clients, web browser, copy-and-paste
- Want to combine local and remote computations seamlessly

Combining capabilities

- For problems requiring combinations of two or more instances of different systems
- Less work than adding capabilities to “home” system
- Even if the “home” system can do it, the “foreign” system may do it much faster!

Parallel computations

- How to exploit multiple CPUs to solve larger problems
- Do this with officially released software as available today

Common limitations

- Interfaces do not work remotely
- Transmission of large or complex objects may be difficult
- To support new CAS, new I/O convertor is needed. It will rely upon the I/O format, may be subject to parsing errors and may be broken by changes in the other CAS
- OpenMath support: not enough deep (i.e. range of CDs and complete syntax/encodings) and wide (i.e. not many CAS)
- Web services: not interactive, just database access
- May not work in some operating systems
- May be difficult for the end-user to customise

SCIEnce
Symbolic Computation Infrastructure for Europe

`http://www.symbolic-computing.org`

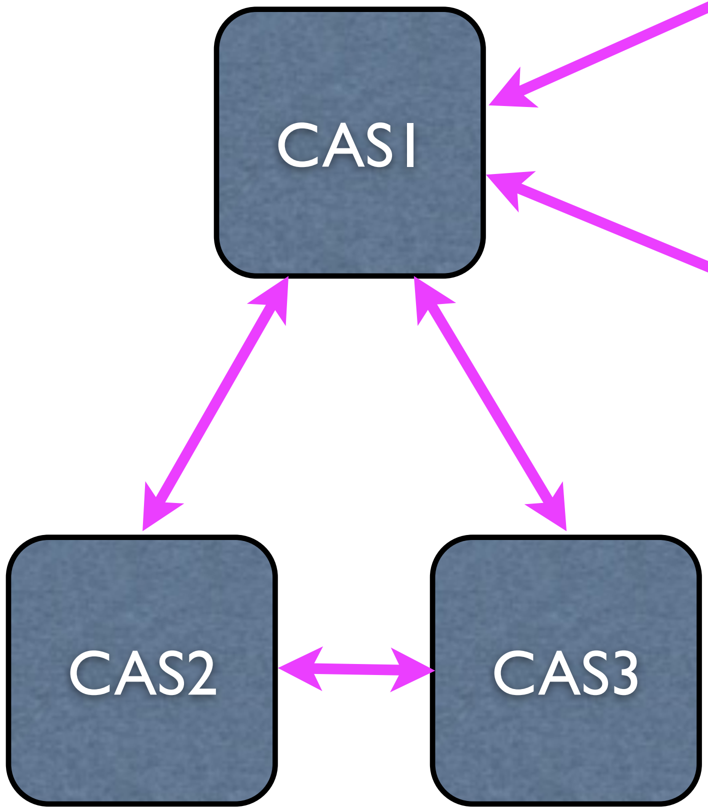
5+ years long research infrastructure project
Framework VI programme grant RII3-CT-2005-026133

- 9 partners
- 7 countries
- 2 continents

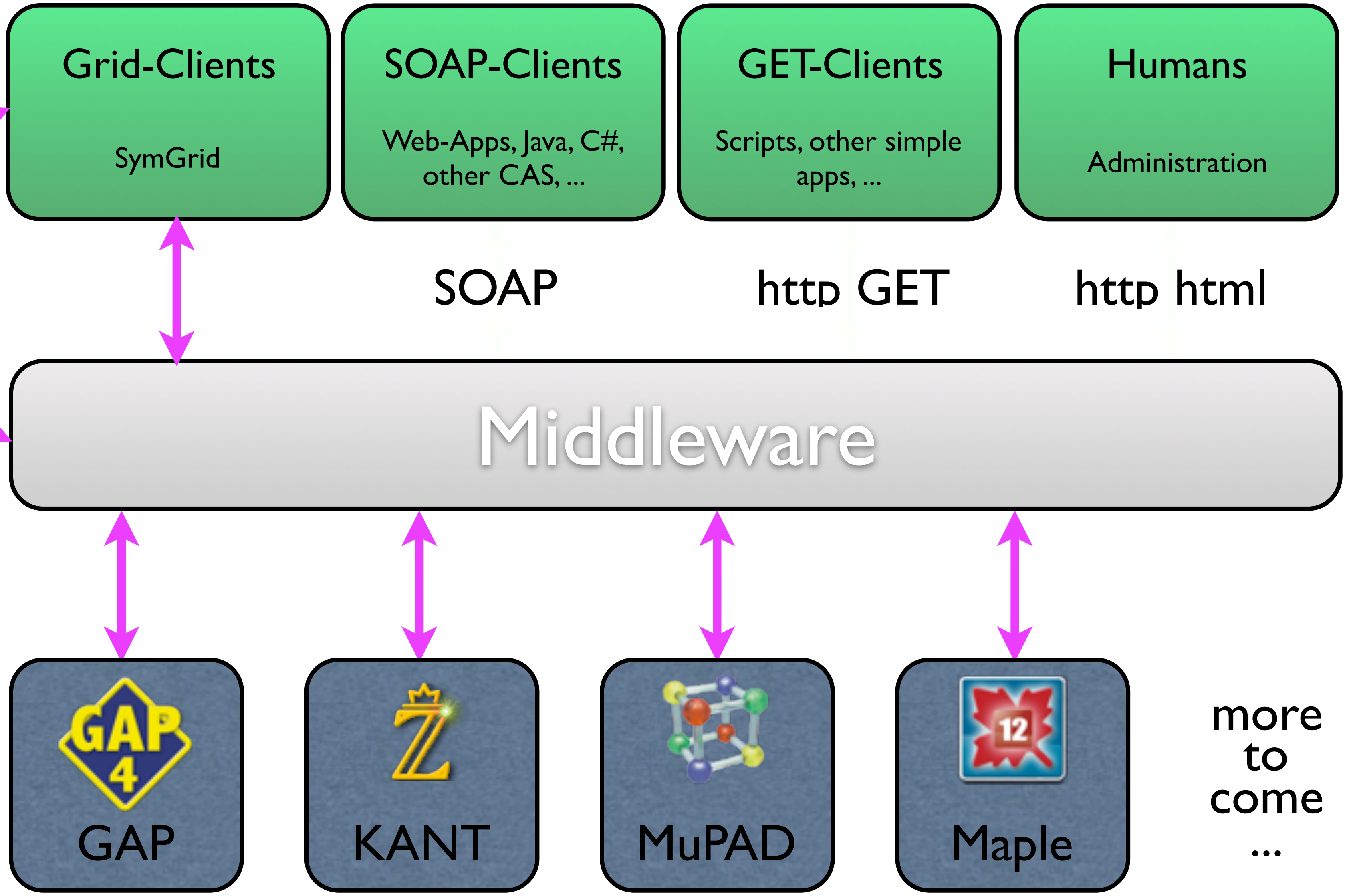
SCSCP

Symbolic Computation Software Composability Protocol

Direct linking CAS to CAS



Linking CAS to other systems



SCSCP

Symbolic Computation Software Composability Protocol

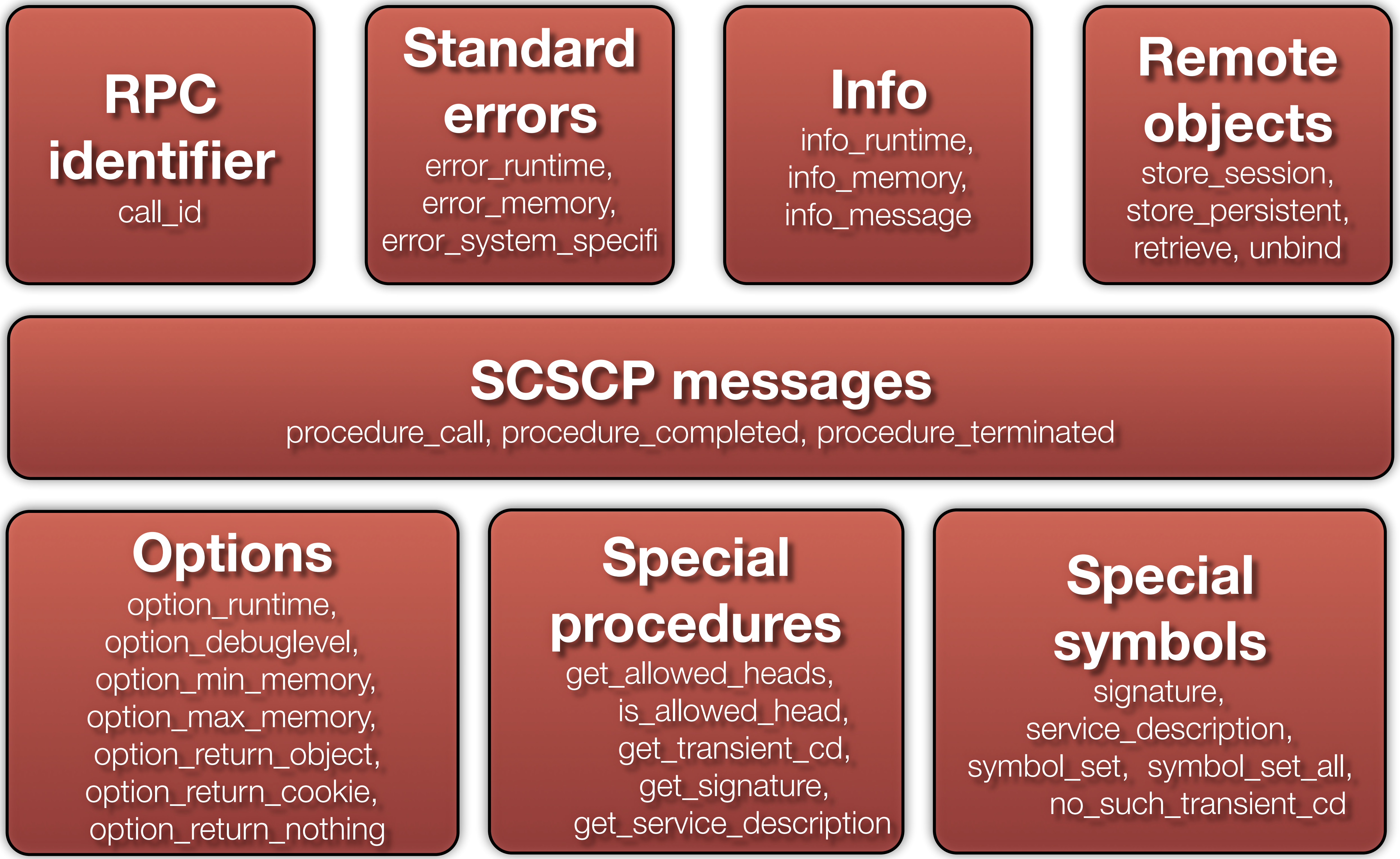
- Remote procedure call protocol for communication between CAS and any other compatible software (another CAS, web-application, etc.)
- SCSCP specification defines messages to and from CAS:
  - procedure call
  - returning result of successfully completed procedure
  - returning a signal about procedure termination
- Both protocol instructions and data encoded in OpenMath
- Implemented within systems rather than in wrappers
- See `http://www.symbolic-computing.org/scscp`

- A standard for representing mathematical objects with respect to their semantics (see http://www.openmath.org)
- Semantics vs presentation: what is <it>S</it><sub>42</sub> ?
  - The Symmetric group of degree 42 ?
  - A sphere in 42-dimensional space ?
  - 1+2+...+42 ?
  - The Answer to the Ultimate Question of Life, The Universe and Everything ???
- Instead, the following OpenMath code means what is says:

```
<OMOBJ>
    <OMA>
        <OMS cd="permgp2" name="symmetric_group"/>
        <OMI>42</OMI>
    </OMA>
</OMOBJ>
```

SCSCP

SYMBOLIC COMPUTATION
SOFTWARE COMPOSABILITY PROTOCOL



RPC identifier

call_id

Standard errors

error_runtime, error_memory, error_system_specifi

Info

info_runtime, info_memory, info_message

Remote objects

store_session, store_persistent, retrieve, unbind

SCSCP messages

procedure_call, procedure_completed, procedure_terminated

Options

option_runtime, option_debuglevel, option_min_memory, option_max_memory, option_return_object, option_return_cookie, option_return_nothing

Special procedures

get_allowed_heads, is_allowed_head, get_transient_cd, get_signature, get_service_description

Special symbols

signature, service_description, symbol_set, symbol_set_all, no_such_transient_cd

GAP implementation of SCSCP

- SCSCP package by AK and Steve Linton
- Included in the GAP distribution
- Provides both client and server functionality
- Uses GAP packages IO (requires compilation on Linux and Mac OS X; Windows binaries are provided with GAP distribution), GAPDoc and OpenMath
- Since GAP 4.5 release both client and server are fully functional on Linux, Mac OS X and Windows
- See http://www.cs.st-andrews.ac.uk/~alexk/scscp/

Simplest example

SCSCP POWERED

- lines from the server configuration file

```
...
InstallSCSCPprocedure( "WS_Factorial", Factorial );
...
RunSCSCPserver("localhost",26133);
```

- The client needs to know the name of the remote procedure, the name of the server and the number of the port

```
gap> EvaluateBySCSCP( "WS_Factorial", [ 12 ], "localhost", 26133 );
rec( attributes := [ [ "call_id", "localhost:26133:12325:GxjuLOvp" ] ],
  object := 479001600 )
```

User-level functionality

- The service provider installs procedures available as SCSCP services and starts the SCSCP server
- The client sends request to the server and gets back result
- This is compatible with any SCSCP-compliant system !!!
- The underlying technology is well-hidden: the end-user may know nothing about OpenMath and SCSCP !!!
- Store/Retrieve procedures allowing to work with remote objects not supported in the native system; objects too large to host them at home system; objects that can not be transmitted or allow only partial transmission with some knowledge that may be lost or too complicated to maintain

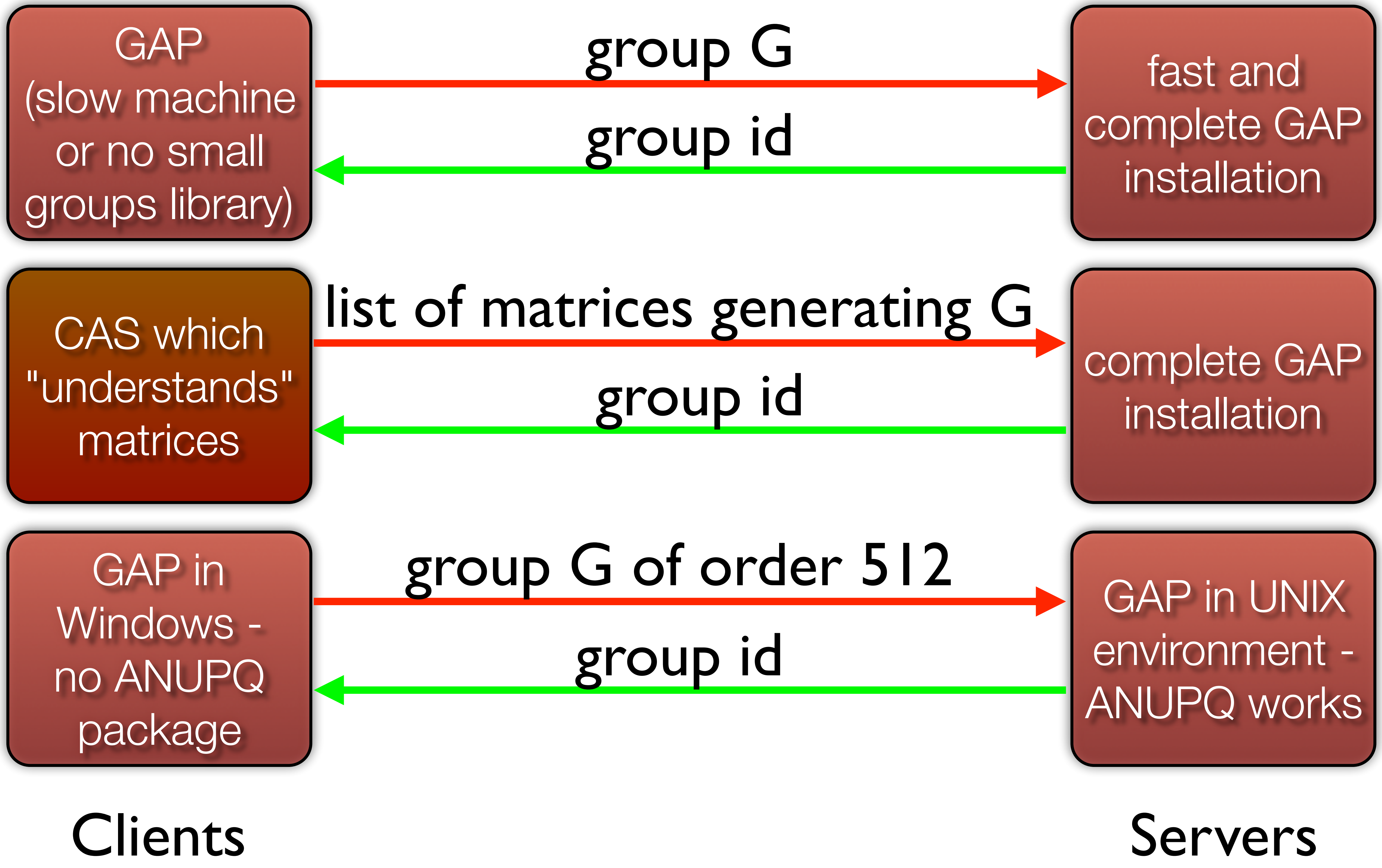
How to configure SCSCP server

- 1. Specify (e.g. in `gap4r6/pkg/scscp/config.g`) setup parameters
- 2. Put all what you need in the configuration file (you may use as a template the file `gap4r6/pkg/scscp/example/myserver.g`):
  - loading all necessary packages and private GAP code
  - installing SCSCP procedures with `InstallSCSCPprocedure("NameForClient", InternalName);`
  - starting the server with `RunSCSCPserver( ... )`
- May control where to listen, whom to answer, what to accept in order to securely provide public SCSCP services
- Start GAP with ‘gap myserver.g’ or as a daemon using the `gap4r6/pkg/scscp/gapd.sh` script (output may be redirected to a file or to /dev/null)

Designing SCSCP services

- The GAP Small Groups Library contains a database of all groups of order up to 2000, except those of order 1024
- For all orders in the database not divisible by 512, groups can be "looked up" to find their number in this library
- For groups of order 512, such lookup is possible with the ANUPQ package
- But ANUPQ does not work under Windows (and may be difficult to compile on some Linux or Mac OS X systems), so we may wish to make the identification of groups of order 512 available as an SCSCP service and call it from GAP sessions on Windows clients

3 approaches to group identification



GAP (slow machine or no small groups library)

group G

group id

fast and complete GAP installation

CAS which "understands" matrices

list of matrices generating G

group id

complete GAP installation

GAP in Windows - no ANUPQ package

group G of order 512

group id

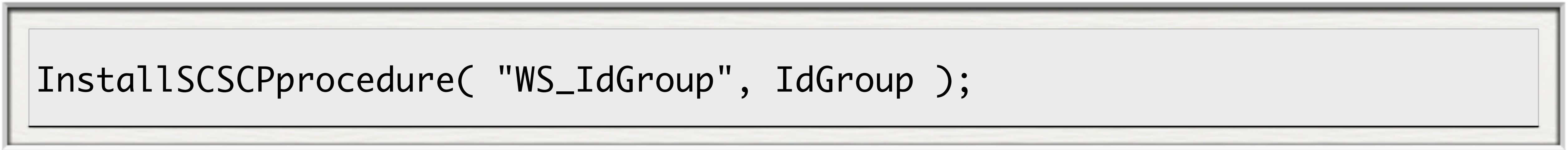
GAP in UNIX environment - ANUPQ works

Clients

Servers

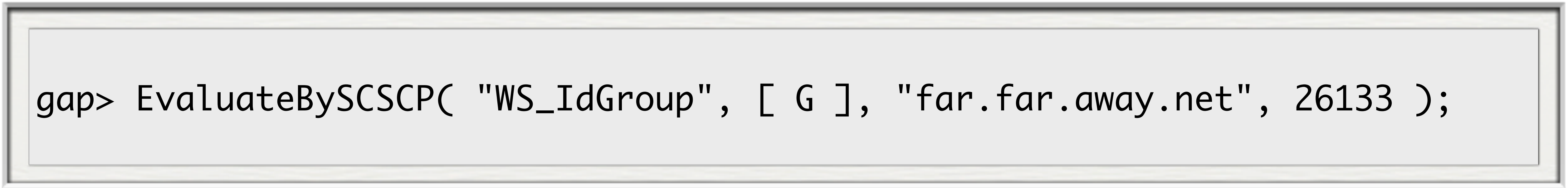
Group -> group id

- Install GAP standard function IdGroup as remotely available procedure



```
InstallSCSCPprocedure( "WS_IdGroup", IdGroup );
```

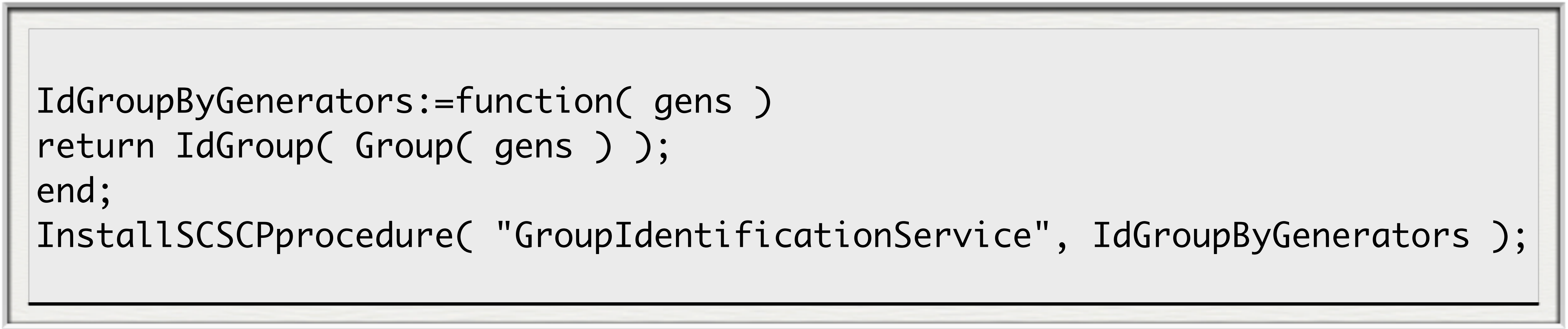
- The client’s call to this procedure will look like



```
gap> EvaluateBySCSCP( "WS_IdGroup", [ G ], "far.far.away.net", 26133 );
```

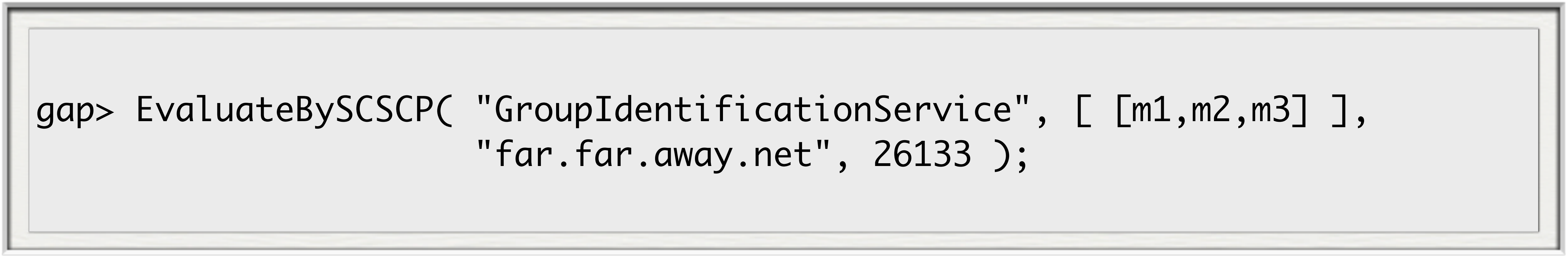
List of matrices -> group id

- Create a function to construct and identify a group generated by these matrices



```
IdGroupByGenerators:=function( gens )
return IdGroup( Group( gens ) );
end;
InstallSCSCPprocedure( "GroupIdentificationService", IdGroupByGenerators );
```

- The client's call to this procedure may look like



```
gap> EvaluateBySCSCP( "GroupIdentificationService", [ [m1,m2,m3] ],
                      "far.far.away.net", 26133 );
```

- Note that errors will be handled automatically

pc-group of order 512 -> group id

- How to encode pc-groups?
- There is no CD for pc-groups (and only a private CD for fp-groups)
- Since we’re only expecting GAP clients, however, we can use a GAP-specific representation – the integer given by `CodePcGroup`
- So our server will offer just one function `IdGroup512ByCode` which will take this number, reconstruct the group from it and return its ID

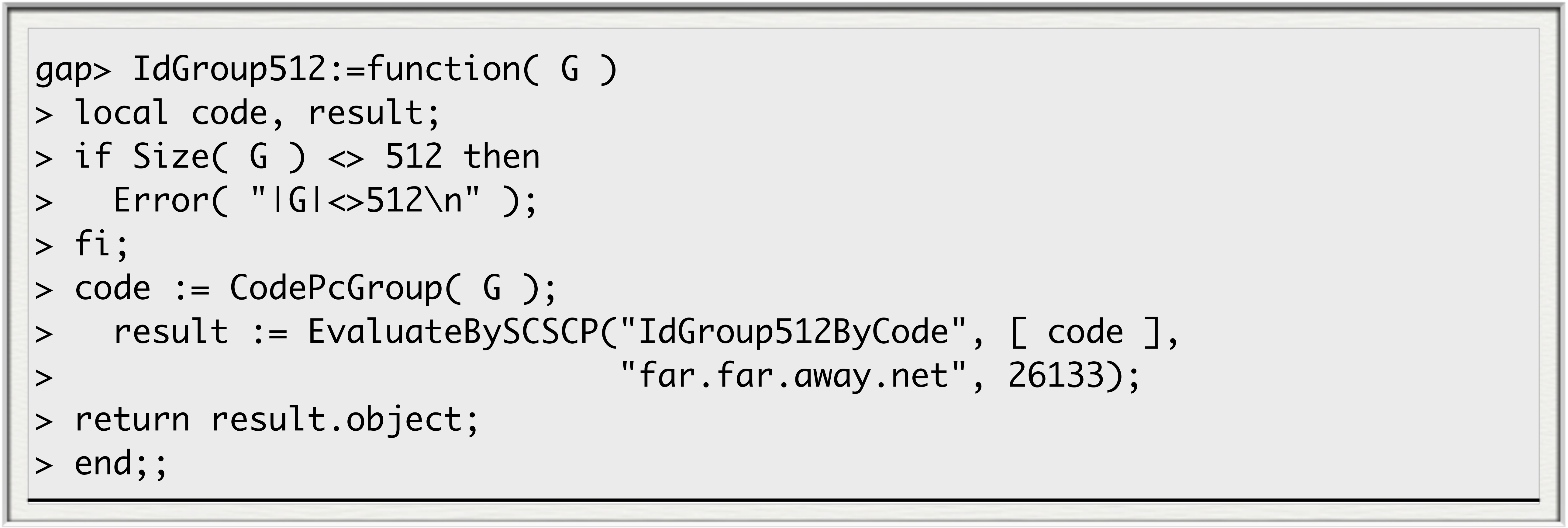
pc-group of order 512 -> group id

- Server-side setup

```
gap> LoadPackage("scscp");; LoadPackage("anupq");;
gap> IdGroup512ByCode := function( code )
> local G, F, H;
> G := PcGroupCode( code, 512 );
> F := PqStandardPresentation( G );
> H := PcGroupFpGroup( F );
> return IdStandardPresented512Group( H );
> end;;
gap> InstallSCSCPprocedure("IdGroup512", IdGroup512ByCode );
InstallSCSCPprocedure : procedure IdGroup512 installed.
gap> RunSCSCPserver( true, 26133 );
```

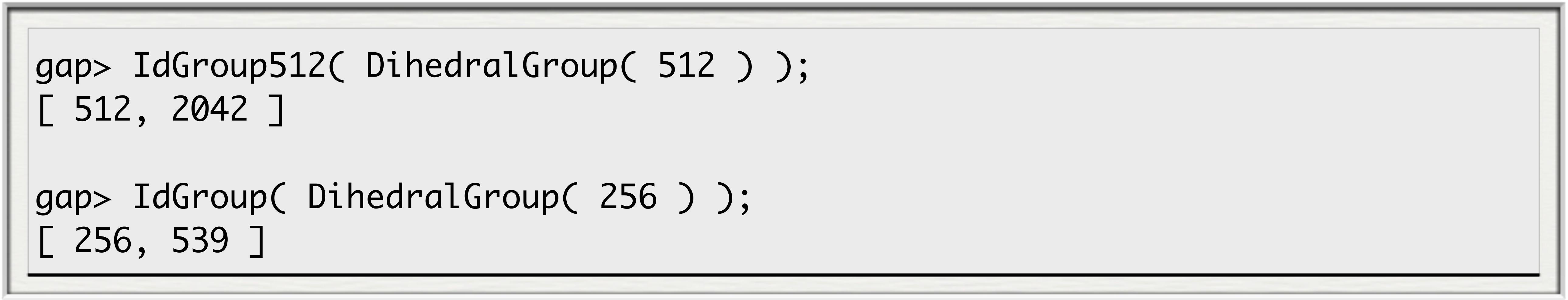
pc-group of order 512 -> group id

- Client-side wrapper



```
gap> IdGroup512:=function( G )
> local code, result;
> if Size( G ) <> 512 then
>   Error( "|G|<>512\n" );
> fi;
> code := CodePcGroup( G );
>   result := EvaluateBySCSCP("IdGroup512ByCode", [ code ],
>                              "far.far.away.net", 26133);
> return result.object;
> end;;
```

- Client-side usage: as user-friendly as standard call to IdGroup



```
gap> IdGroup512( DihedralGroup( 512 ) );
[ 512, 2042 ]

gap> IdGroup( DihedralGroup( 256 ) );
[ 256, 539 ]
```

- Is this limited to functionality/data types for which CDs exist ?
  - Avoid this by allowing *transient* CDs, which contain symbols specific to that service, obtainable from the server on request
- Encoding may be unreasonably bulky, or encoding costs may be too high for some applications
  - Perfectly OK for services to pass data in some private format encoded in a *private* CD or using OMSTRING, OMBYTES or OMFOREIGN element, if that suits the application.
- Both transmission of actual mathematical objects and *references* to them are supported
- New CD may be designed for efficient representation if the standard CD is not enough (e.g. matrices over finite fields)

Ways to run parallel computations in GAP

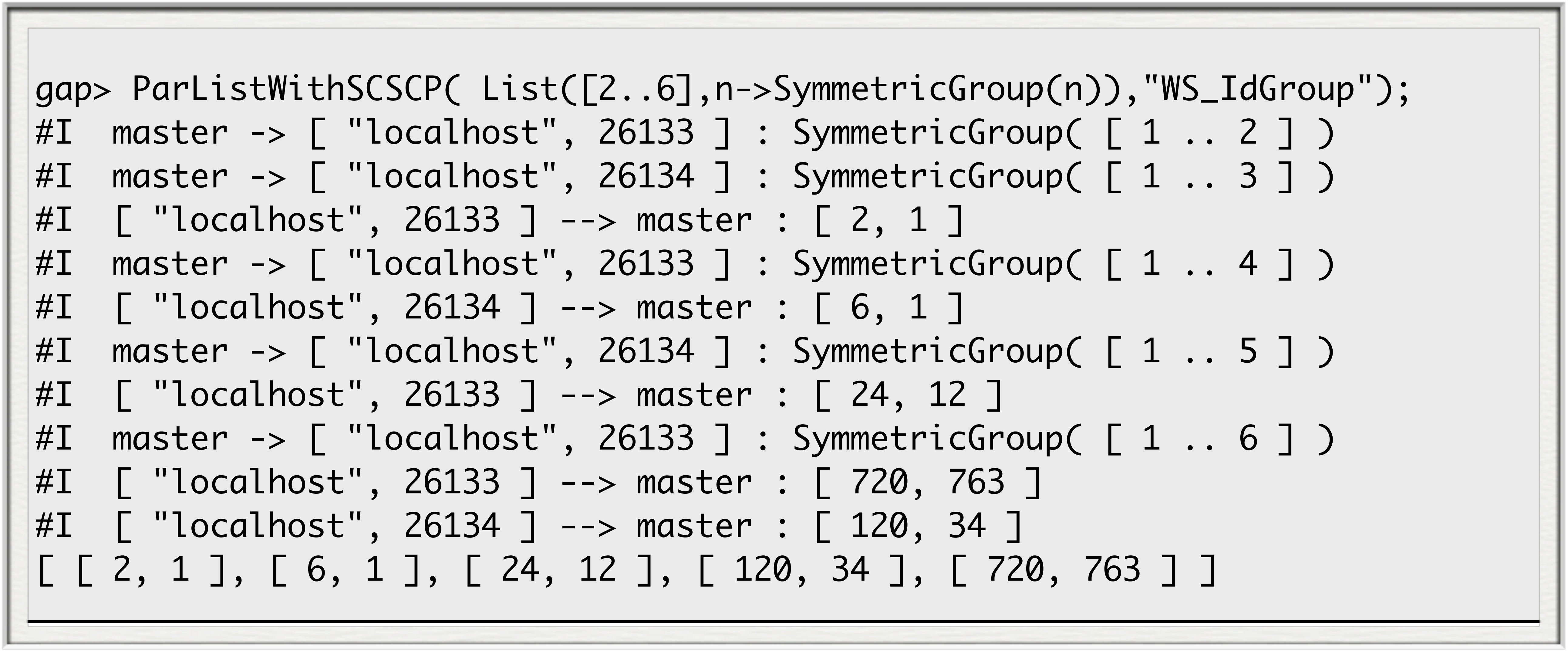
- Traditional job submission systems (PBS, Condor)
- In the current release of GAP also with the ParGAP package using MPI (Message Passing Interface)
- HPC-GAP alpha-release (http://www-circa.mcs.st-and.ac.uk/hpcgap.php):
  - shared memory programming model using threads
  - distributed memory programming model using MPI
- But what can you do only in GAP, avoiding external binaries as much as possible?
- For example, to create an “ad hoc” cluster from several computers

Parallel computing with SCSCP

- Issuing multiple remote procedure calls
- Waiting till all of them will be completed
- Waiting for the first available result and discarding the rest
- Implemented in GAP : easy to learn and modify
- Master-Worker skeleton on top of this

Parallel computations with SCSCP

- Master-worker skeleton

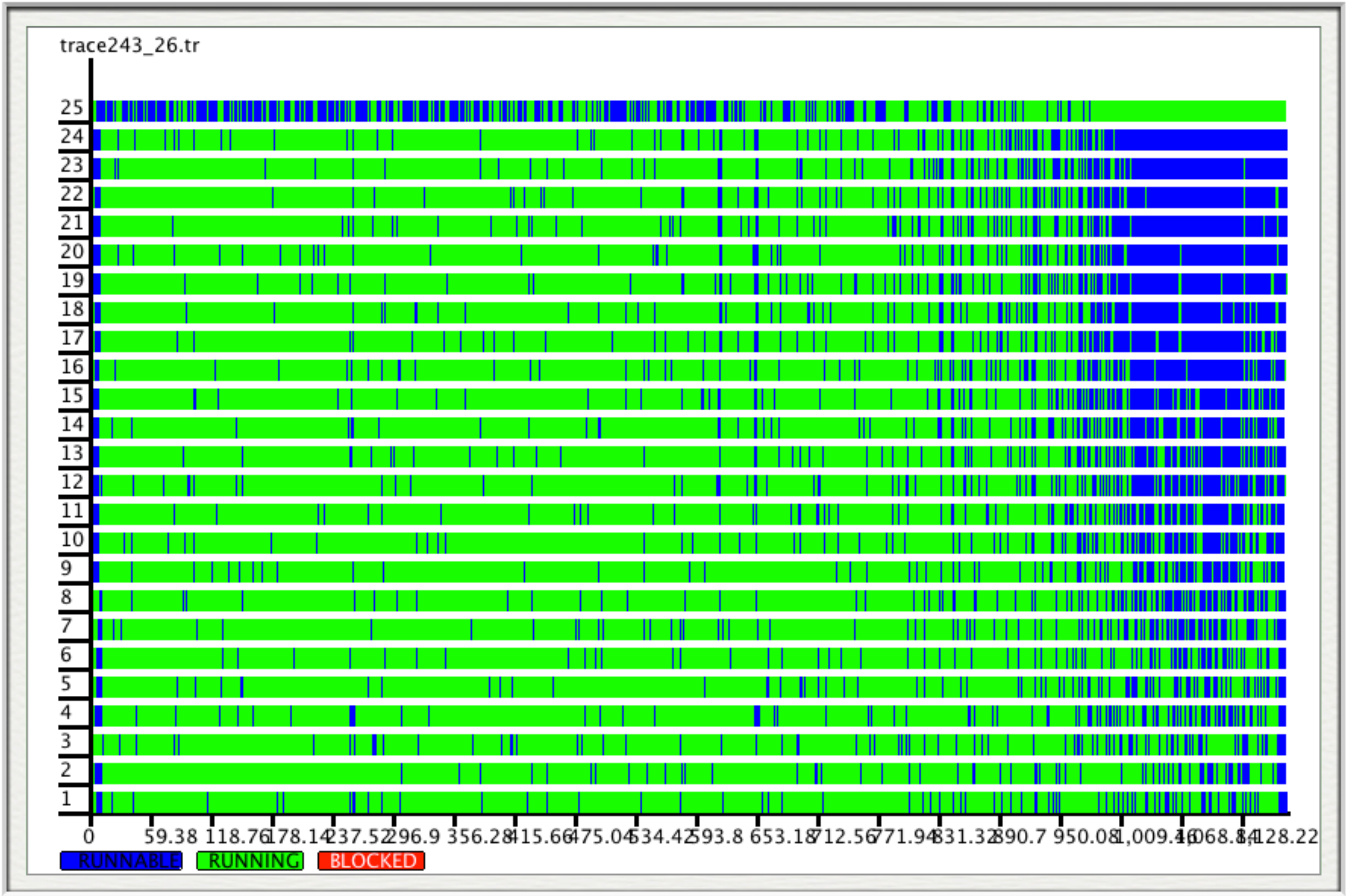


```
gap> ParListWithSCSCP( List([2..6],n->SymmetricGroup(n)),"WS_IdGroup");
#I  master -> [ "localhost", 26133 ] : SymmetricGroup( [ 1 .. 2 ] )
#I  master -> [ "localhost", 26134 ] : SymmetricGroup( [ 1 .. 3 ] )
#I  [ "localhost", 26133 ] --> master : [ 2, 1 ]
#I  master -> [ "localhost", 26133 ] : SymmetricGroup( [ 1 .. 4 ] )
#I  [ "localhost", 26134 ] --> master : [ 6, 1 ]
#I  master -> [ "localhost", 26134 ] : SymmetricGroup( [ 1 .. 5 ] )
#I  [ "localhost", 26133 ] --> master : [ 24, 12 ]
#I  master -> [ "localhost", 26133 ] : SymmetricGroup( [ 1 .. 6 ] )
#I  [ "localhost", 26133 ] --> master : [ 720, 763 ]
#I  [ "localhost", 26134 ] --> master : [ 120, 34 ]
[ [ 2, 1 ], [ 6, 1 ], [ 24, 12 ], [ 120, 34 ], [ 720, 763 ] ]
```

Parallel computations with SCSCP

Communication layer	SCSCP
Environment	Linux, Mac OS X, Windows - anything where SCSCP client/ server works
Supported workers	any SCSCP-compliant CAS
Heterogeneity	No limits on operating system, architecture, location
Fault-tolerance	Retrying on another worker Adding new worker
Even more	More complex networks, timeouts, shared structures ...

Profiling with EdenTV:
(master, 8 local workers and 2x8 remote workers)



Normalised unit group of a modular group algebra: the result is a group of order 3^242

Computed sequentially: 5 hr 8 min, in parallel: 19 m 31 sec. Speedup 15.92

SCSCP

SYMBOLIC COMPUTATION
SOFTWARE COMPOSABILITY PROTOCOL

Implementations as on today

- GAP, KANT, MuPAD (currently inside MATLAB), Maple
- Even more: Mathematica, Macaulay2 (out of box), TRIP (out of box), Coq (prototype), Magma (wrapper), ...
- Java OpenMath and SCSCP API: `java.symcomp.org`
- A collection of tools and prototypes that were built around this API (WUPSI, ISS, LattViz, SkySym, ...)
- C/C++ API that originated from SCSCP support in TRIP
- MiniSCSCP++ (a C++ library with a simple C++ client)
- A simple SCSCP client written in Python

Further details

- SCSCP specification
- Manuals for corresponding SCSCP-compliant CAS extensions
- *"Easy composition of symbolic computation software using SCSCP: A new Lingua Franca for symbolic computation"* by S.Linton, K.Hammond, AK, C.Brown, P.W.Trinder, H.-W.Loidl, P.Horn and D.Roozemond, J. Symbolic Computation 49 (2013), 95-119
- *"Parallel computations in modular group algebras"* by AK and S.Linton, Proceedings of PASCO 2010 (Grenoble, July 21-23, 2010): case study and tutorial on optimising the parallel performance in our model
- *"The modular isomorphism problem for the groups of order 512"* by B.Eick and AK, Proceedings of Groups St Andrews in Bath 2009, Cambridge University Press