



IBM Rochester

BGP OSS — Revision 2.3 Messaging OSS plan

DCMF Messaging team

BlueGene Comm Team

19-Feb-08

© 2007 IBM Corporation



IBM Rochester

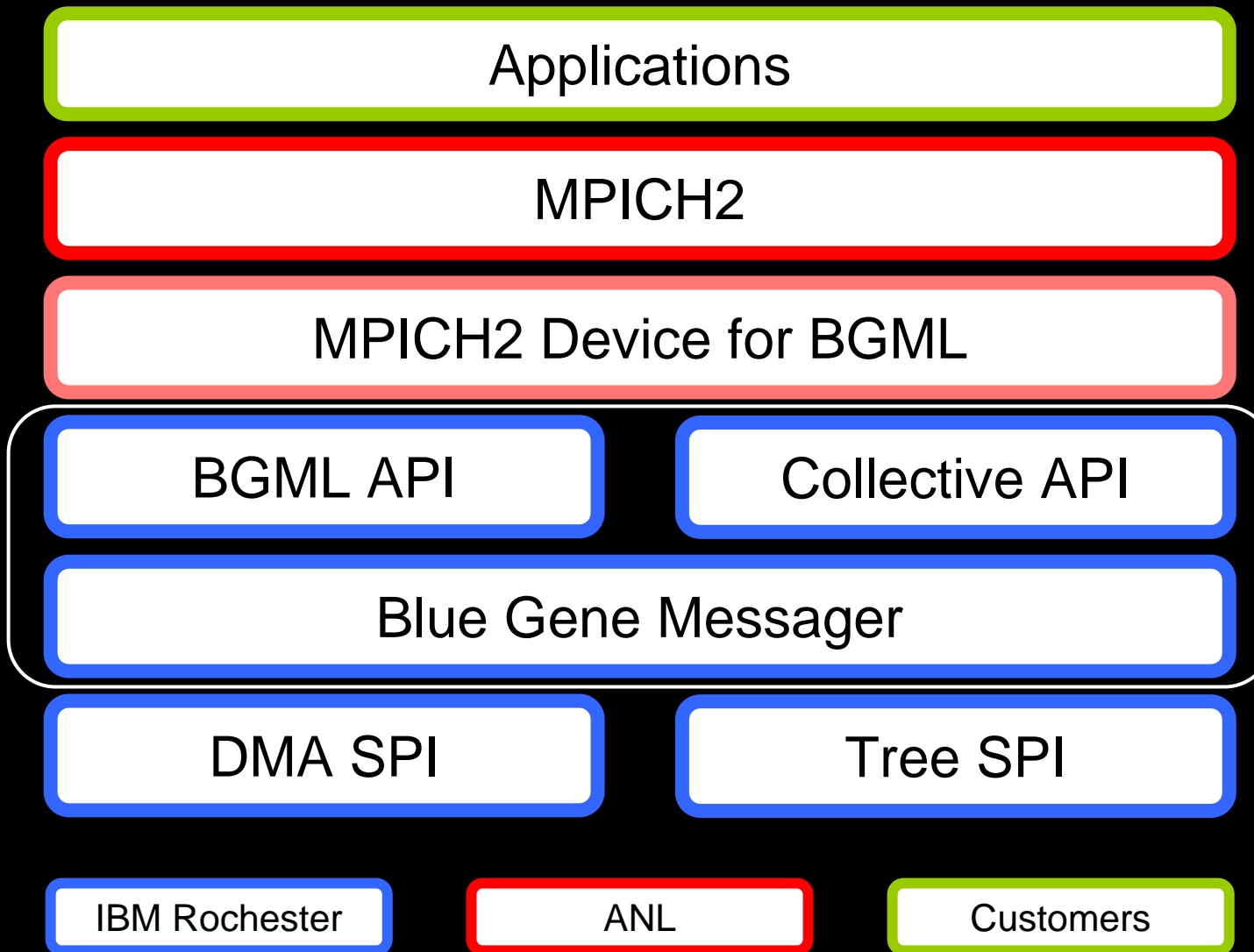
BGP OSS [Part 1/2] Source Organization

BlueGene Comm Team

19-Feb-08

© 2007 IBM Corporation

BG Comm stack



Contribution expectations

- **MPICH2 and associated Device**

- ANL currently maintains MPICH2, and we would like them to maintain our device as well. Even if they don't, we do not expect significant contributions.

- **Messaging and the APIs**

- We (IBM Rochester) plan to maintain these, and we expect quite a number of contributions.

- **SPIs**

- These are so low-level that we do not expect much more than bug reports.

Two approaches

- **“Traditional” open source model**
 - This would be used for the SPIs and MPICH2.
 - There is exactly one code base, and it is meant to be stable.
 - Though not always stable at the bleeding edge, it would certainly be stable by release time.
- **“Split” open source model**
 - This would be used for the messaging and collectives.
 - There is one code base, but it is split into several modules.
 - Some modules are production versions that have been stabilized, while others may be contributed versions that are still under development.

The parts of Messaging

- **Modules in the Messaging library**
 - Devices: implement a network (i.e. tree, torus, GI, dma, etc)
 - Messenger: implements parts of the common messaging API
 - Protocols: implement communication protocols
 - Sysdeps: system dependencies (platform support)
 - Tests: ensure the quality of the code produced
- **Non prod/contrib modules**
 - Logging
 - Math
 - Queuing

The parts of Collectives

- **Modules in the Collectives library**
 - This library does not yet have a strong prod/contrib directory layout.
 - Adaptor
 - Kernel
 - Tests
- **Non prod/contrib modules**
 - Docs
 - Tools

A closer look at the “Split” model

- **Each of the major topics contains two sections**
 - prod/ Code supported and shipped by IBM
 - contrib/ Free area for submissions
 - contrib/>{site} The contrib/ area is divided by site for ease of use



IBM Rochester

BGP OSS [Part 2/2]

Code Submission Process

BlueGene Comm Team

19-Feb-08

© 2007 IBM Corporation

A proposal for collaboration

- **The IBM Rochester comm team will fulfill two roles:**
 1. Product delivery and all requirements associated with SOW delivery.
 2. OSS maintainer for the source code tree, encouraging external collaboration.
 - Maintainer has full write access to the tree;
 - Patches to prod code go through the master maintainer.
 - Patches to contrib code are simply accepted.
- **IBM Research, ANL, and other customers are external collaborators.**
 - Each site will have a known site maintainer.

External → IBM Rochester relations

- **Each site maintainer will be responsible for all interaction from that site.**
- **That site will be assigned an internal *Patch Butler* (PB) on the IBM Rochester comm team who will be their point of contact.**
 - The PB is the person who handles the “*prod/ patch approval*”, ensuring that the work moves along without getting lost.
 - Patches need to be submitted via the mail list.
- **We claim a concept of local repository versus primary repository:**
 - Local repository: a user’s files as created by a “`git clone ..`”. Clearly, a user may do anything in this sandbox, including changes to *prod/*. The patch process only concerns code which is to be pushed to the primary.
 - Primary repository: the central backing store. Changes made here can be seen by all users, and should only be made to further the aims of the project. This is also known as the “external *prod*” repository.

Prod/ Patch Approval (how to officially change prod/)

- **PB receives update request and patch/file list**
- **PB handles basic sanity checking individually. It must have:**
 - Expert approval—possible if the PB doesn't follow that code.
 - Team approval—probably needed. This is likely a code review with people selected by PB.
 - Business case discussion—needed if a release is quickly approaching.
 - Test team approval—the code must pass the test bucket.
- **Commit to Git**

Prod/ Patch Approval [Detailed]

- **PB receives update request and patch/file list**
- **PB handles basic sanity checking individually. It must have:**
 - COO (or whatever lawyers tell us) & proper copyright/license headers
 - Explanation of how to update the repository
 - Updated build target in the make system to build the new source
 - Sufficient documentation in updated files
 - Test cases for functionality/perf exist
 - Before/After tests show what was deficient but is now better
 - Ensure that there are no references to contrib/
 - Successful build in clean tree with all contrib/ source code deleted

```
find -depth -name contrib -exec rm -rf {} \;
```
- **Expert approval**
 - The person on the team most familiar with the code/area in question reviews the code to make sure it isn't clearly deficient
- **Team approval**
 - Check code cleanliness & documentation
 - Check that code is maintainable by future contributors (internal & external)
- **Business case discussion (seldom)**
 - As official releases get closer, more complicated patches are less likely to be approved
- **Test team approval**
 - Especially in the case of large patches, more advanced testing by the test team may be required.
- **Commit to Git**

Prod/ change case studies

- **From the perspective of a developer/contributor, the following slides show how to make changes to the source code and contribute the changes to the prod/team.**
 - Checking source into contrib/ does not have these sorts of requirements or process (except “Don’t clobber code committed by others”)
- **The directions on the following slides are suggestions for the developer, unless specifically attributed to the PB.**
- **All contributions should be vetted by the site maintainer before being submitted.**

Prod/ change case study : Small change to single part

- **Make changes and test in local space.**
- **Patch and associated material are submitted to PB.**
- **PB manages the approval process, returning either an “Accept” or a “Rejection”. In the latter case, an explanation will also be forthcoming.**
- **This is the simplest case, and is correspondingly the easiest to get approved.**

Prod/ change case study : Large change to single part

- **Note : “Large” is a relative term, and a user may choose to use the previous method even for larger changes to existing parts.**
- **Copy the local prod/ version to contrib/ repository, possibly using a different name.**
- **Make changes and test in contrib/ repository.**
- **Tarball/file locations and associated material are submitted to PB.**
- **PB manages the approval process, returning either an “Accept” or a “Rejection”. In the latter case, an explanation will also be forthcoming.**
- **The approval difficulty is positively related to the amount of change involved.**

Prod/ change case study : Single new part

- **This case exists when adding a completely new part or redesigning one in a way that the previous versions aren't relevant.**
- **Make changes and test in contrib/ repository.**
- **Tarball/file locations and associated material are submitted to PB.**
- **PB manages the approval process, returning either an "Accept" or a "Rejection". In the latter case, an explanation will also be forthcoming.**

Prod/ change case study : Large change to many parts OR Many new parts

- **This case exists when adding many completely new parts or completely redesigning significant chunks of a messenger.**
- **Make changes and test in contrib/ repository.**
- **Tarball/file locations and associated material are submitted to PB.**
- **PB manages the approval process, returning either an “Accept” or a “Rejection”. In the latter case, an explanation will also be forthcoming.**
- **It is safe to say that this is a *very* difficult one to accept into prod—probably requiring a good business case. If eventual inclusion in prod/ is a requirement, it is probably best to discuss and design the new work with the IBM Rochester team.**

Recommendations for OSS infrastructure

- **These are currently suggestions for servers and such**
 - A source code management system
 - Git—see SCM doc
 - Public (or semi-public) Wiki
 - Internally, we've found that MediaWiki provides an excellent SW platform on which to create a collaborative documentation and repository.
 - This would allow users to create documentation without actually requiring a person or team dedicated to the task.

Recommendations for OSS infrastructure

- **These are currently suggestions for servers and such**
 - Public bug tracking tool (E.g. Bugzilla)
 - Some method of tracking bugs/work items/features will be required. Lotus Notes DBs don't seem to make sense, since external users would not have access.
 - Mailing lists (E.g. Mailman)
 - To ensure maximum levels of communication, it is important that we have a mailing list.
 - To ensure transparency, it is best that all patches be submitted via the mailing list.
 - Emergency contacts
 - These servers form an important system—the electronic backbone of the project.