QEMU interface introspection: From hacks to solutions

Markus Armbruster <armbru@redhat.com>

KVM Forum 2015

Part I
What's the problem?

Interfacing with QEMU

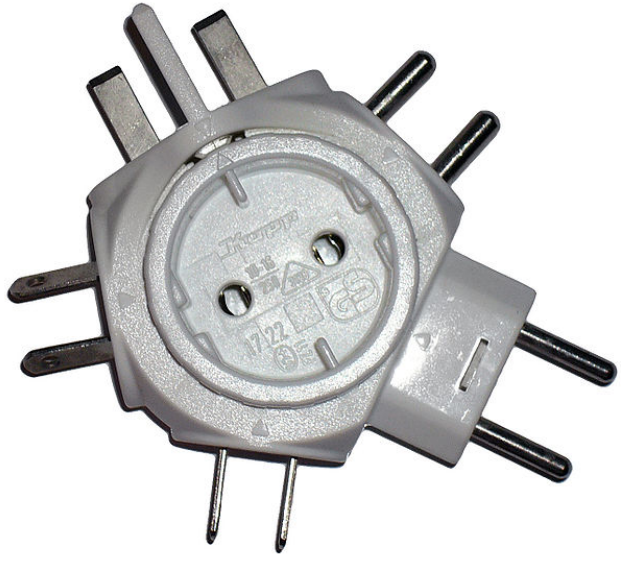
QEMU provides interfaces

- QMP Monitor
- Command line

to management applications like libvirt

QEMU evolves rapidly $\rightsquigarrow$

Many interface versions



Our command line is big

In v2.4:

139 total options

-14 deprecated

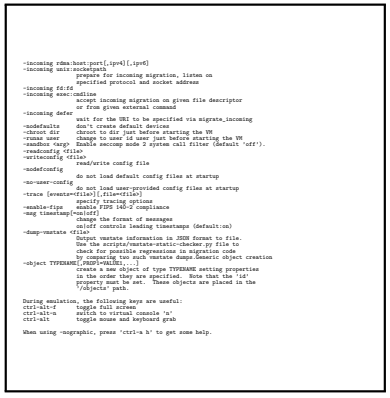
-2 internal use

---

123 supported options

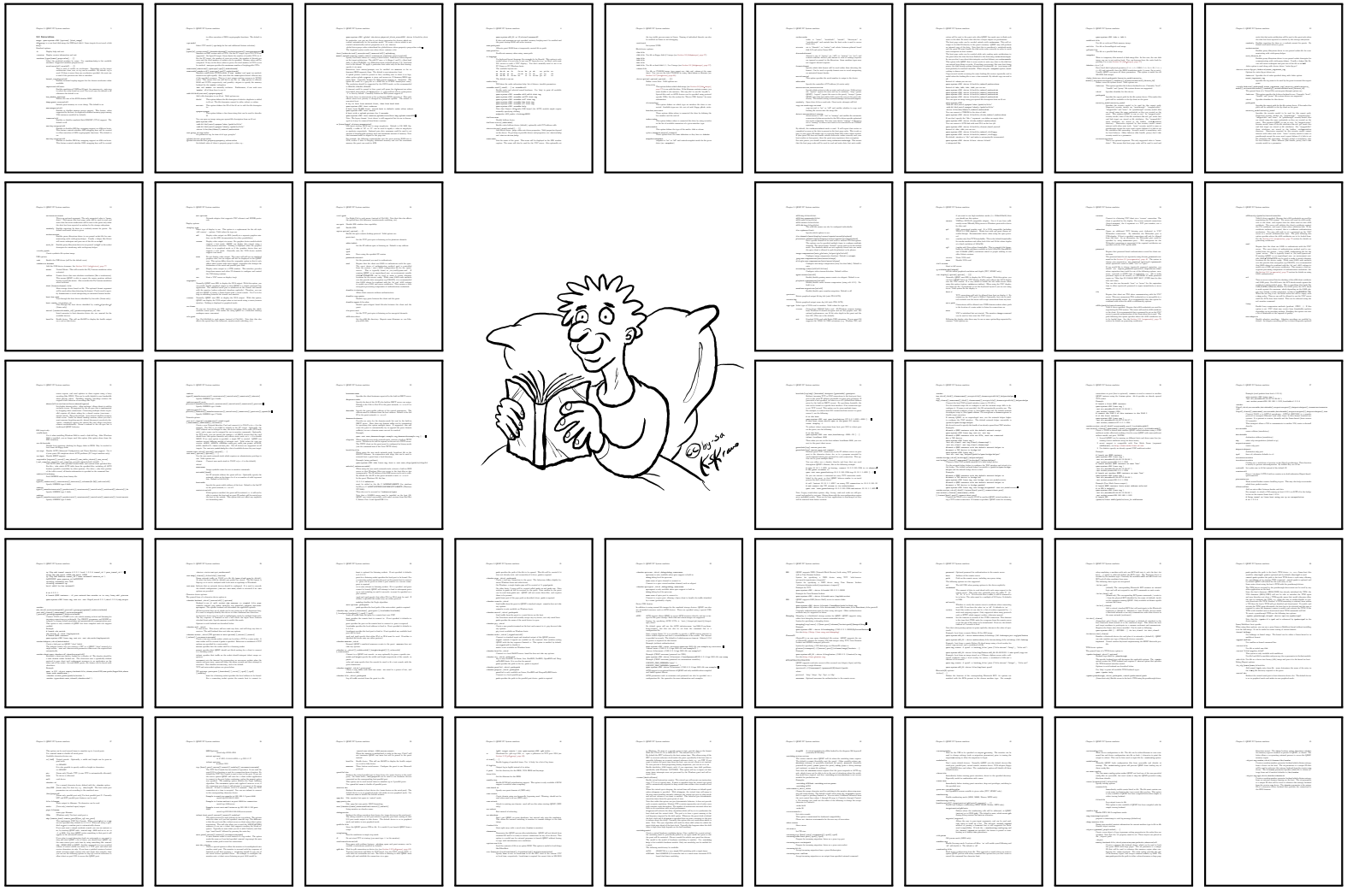
If I had a coin for each of them...

It’s big: output of -help



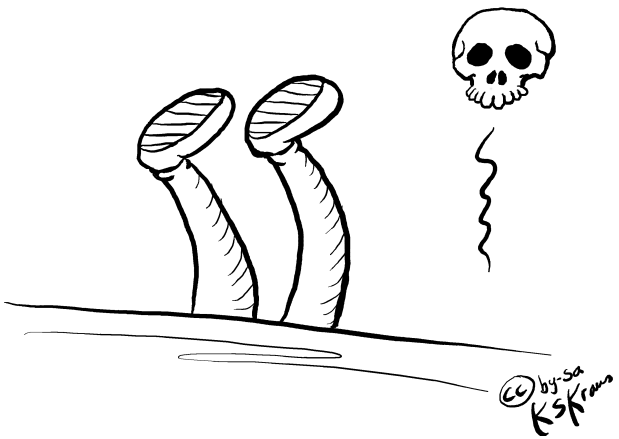
redhat

Really big: its manual section



This is how it's used in anger

```
$ /usr/bin/qemu-system-x86_64 -machine accel=kvm -name boxes-unknown -S -machine pc-i440fx-1.6,accel=kvm,usb=off -cpu \
Penryn -m 3115 -realtime mlock=off -smp 4,sockets=1,cores=4,threads=1 -uuid 8bd53789-adab-484f-8c53-a6df9d5f1dbf -no-u\
ser-config -nodefaults -chardev socket,id=charmonitor,path=/home/guillaume/.config/libvirt/qemu/lib/boxes-unknown.moni\
tor,server,nowait -mon chardev=charmonitor,id=monitor,mode=control -rtc base=utc,driftfix=slew -global kvm-pit.lost_ti\
ck_policy=discard -no-shutdown -global PIIX4_PM.disable_s3=1 -global PIIX4_PM.disable_s4=1 -boot strict=on -device ich\
9-usb-ehci1,id=usb,bus=pci.0,addr=0x5.0x7 -device ich9-usb-uhci1,masterbus=usb.0,firstport=0,bus=pci.0,multifunction=o\
n,addr=0x5 -device ich9-usb-uhci2,masterbus=usb.0,firstport=2,bus=pci.0,addr=0x5.0x1-device ich9-usb-uhci3,masterbus=u\
sb.0,firstport=4,bus=pci.0,addr=0x5.0x2 -device virtio-serial-pci,id=virtio-serial0,bus=pci.0,addr=0x6 -device usb-cci\
d,id=ccid0 -drive file=/home/guillaume/.local/share/gnome-boxes/images/boxes-unknown,if=none,id=drive-ide0-0-0,format=\
qcow2,cache=none -device ide-hd,bus=ide.0,unit=0,drive=drive-ide0-0-0,id=ide0-0-0,bootindex=1 -drive if=none,id=drive-\
ide0-1-0,readonly=on,format=raw -device ide-cd,bus=ide.1,unit=0,drive=drive-ide0-1-0,id=ide0-1-0 -netdev tap,fd=23,id=\
hostnet0 -device rtl8139,netdev=hostnet0,id=net0,mac=52:54:00:db:56:54,bus=pci.0,addr=0x3 -chardev spicevmc,id=charsma\
rtcard0,name=smartcard -device ccid-card-passthru,chardev=charsmartcard0,id=smartcard0,bus=ccid0.0 -chardev pty,id=cha\
rserial0 -device isa-serial,chardev=charserial0,id=serial0 -chardev spicevmc,id=charchannel0,name=vdagent -device virt\
serialport,bus=virtio-serial0.0,nr=1,chardev=charchannel0,id=channel0,name=com.redhat.spice.0 -device usb-tablet,id=in\
put0 -spice port=5901,addr=127.0.0.1,disable-ticketing,image-compression=off,seamless-migration=on -device qxl-vga,id=\
video0,ram_size=67108864,vram_size=67108864,vgamem_mb=16,bus=pci.0,addr=0x2 -device AC97,id=sound0,bus=pci.0,addr=0x4 \
-chardev spicevmc,id=charredir0,name=usbredir -device usb-redir,chardev=charredir0,id=redir0 -chardev spicevmc,id=char\
redir1,name=usbredir -device usb-redir,chardev=charredir1,id=redir1 -chardev spicevmc,id=charredir2,name=usbredir -dev\
ice usb-redir,chardev=charredir2,id=redir2 -chardev spicevmc,id=charredir3,name=usbredir -device usb-redir,chardev=cha\
rredir3,id=redir3 -incoming fd:20 -device virtio-balloon-pci,id=balloon0,bus=pci.0,addr=0x7 -msg timestamp=on
```

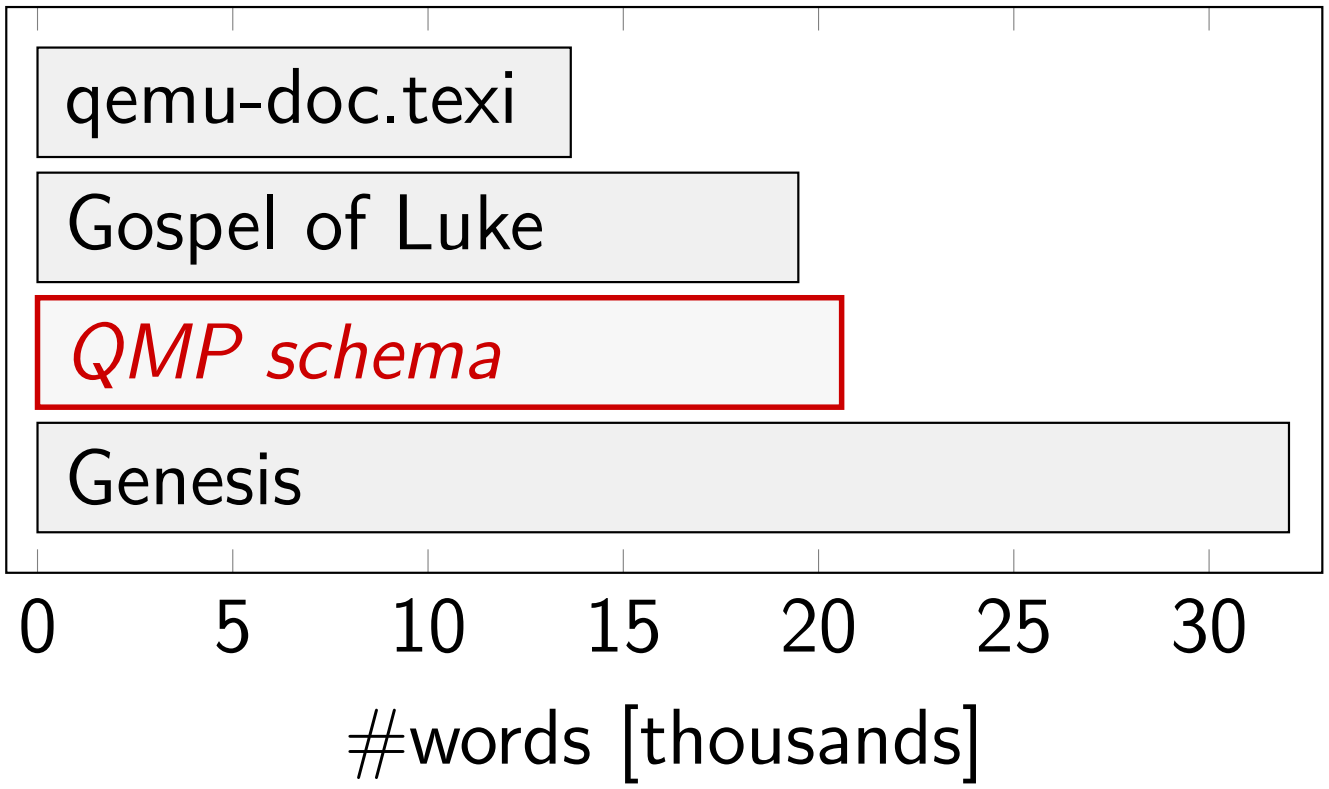


QMP is *even bigger*

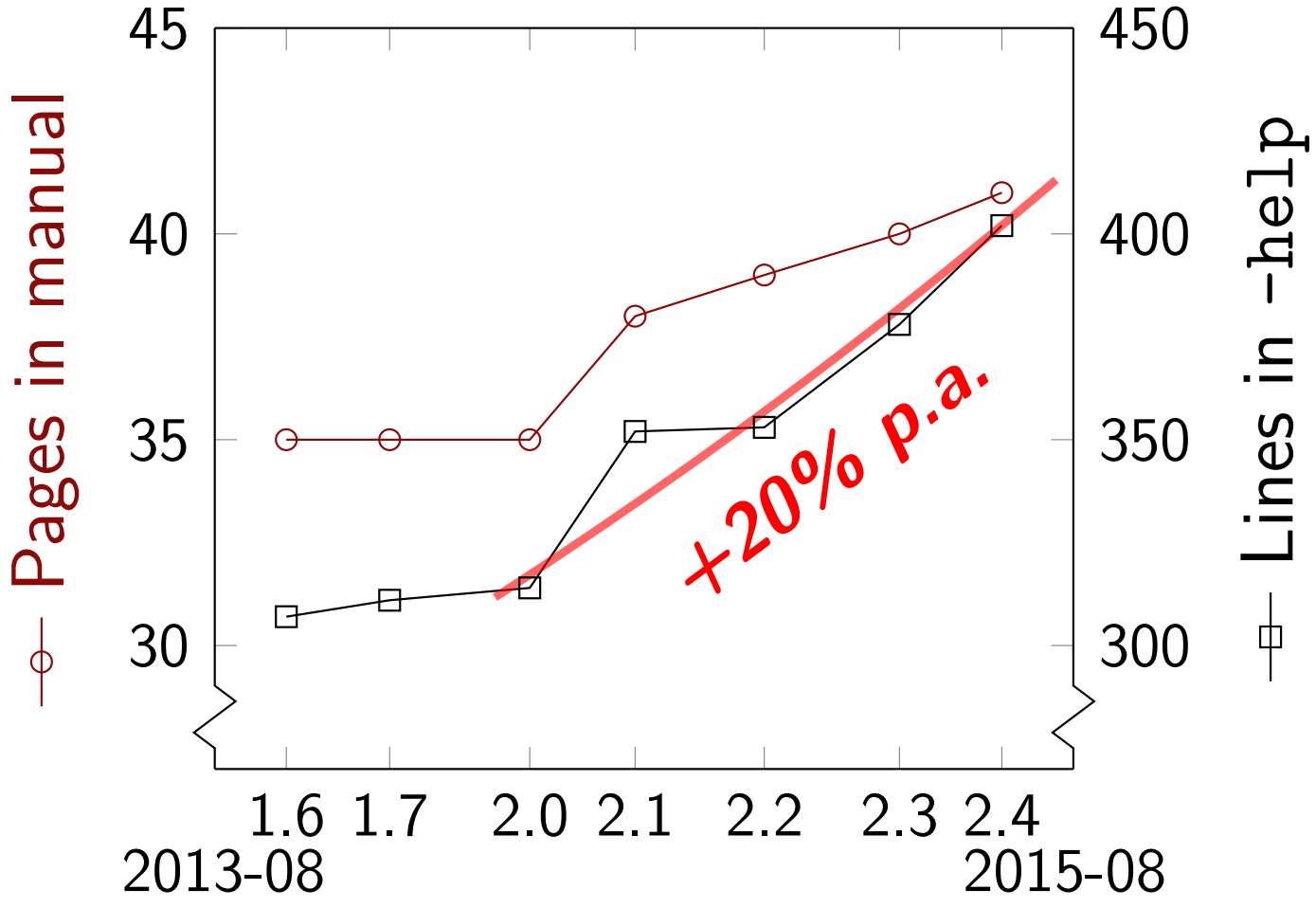
126 commands + 33 events

More than 700 named arguments and results

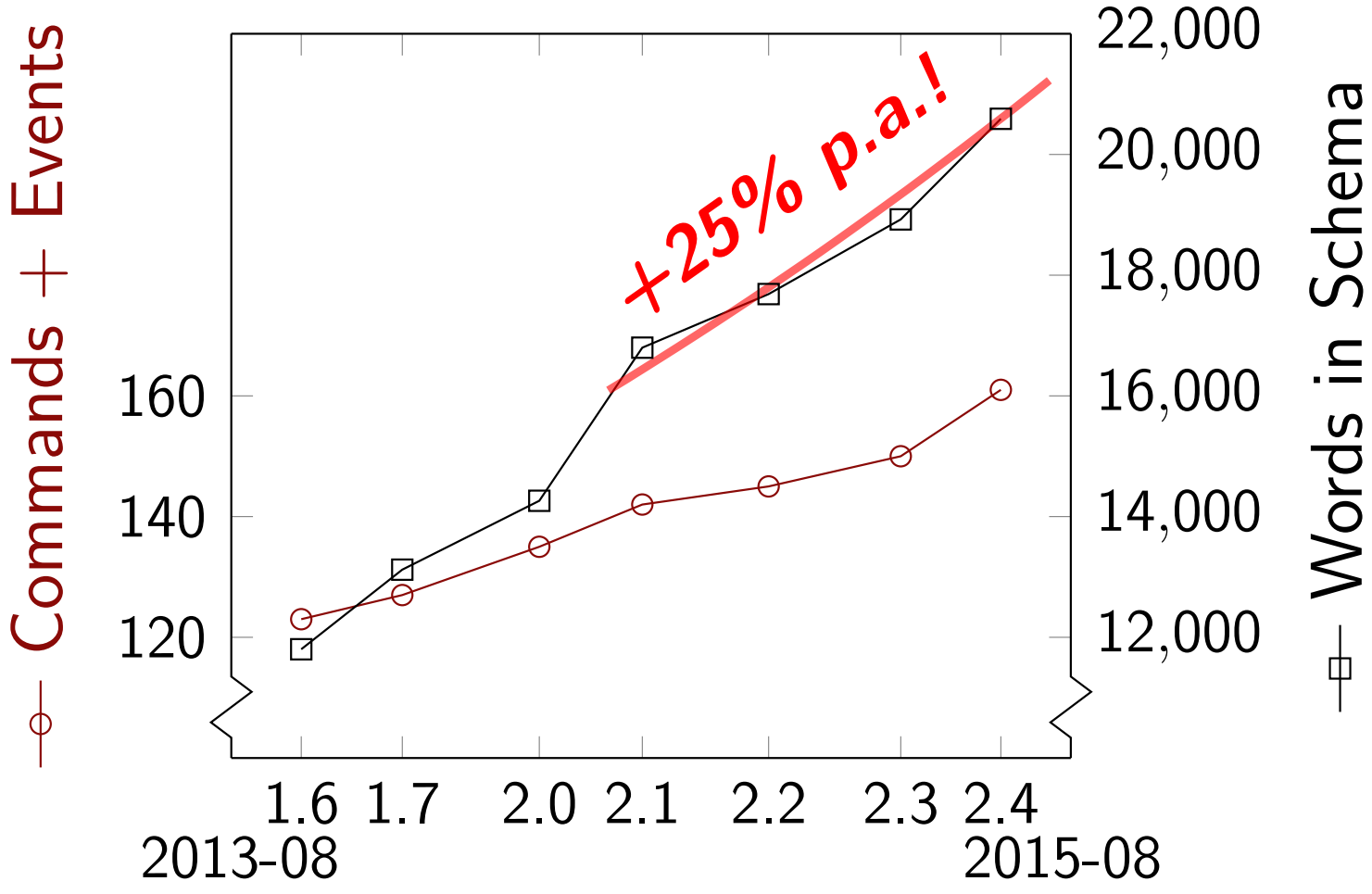
Defined in the (book-sized) QAPI/QMP schema



Command line evolves fast



QMP evolves *faster*



Why interface introspection?

QEMU provides big, rapidly evolving interfaces

A program can

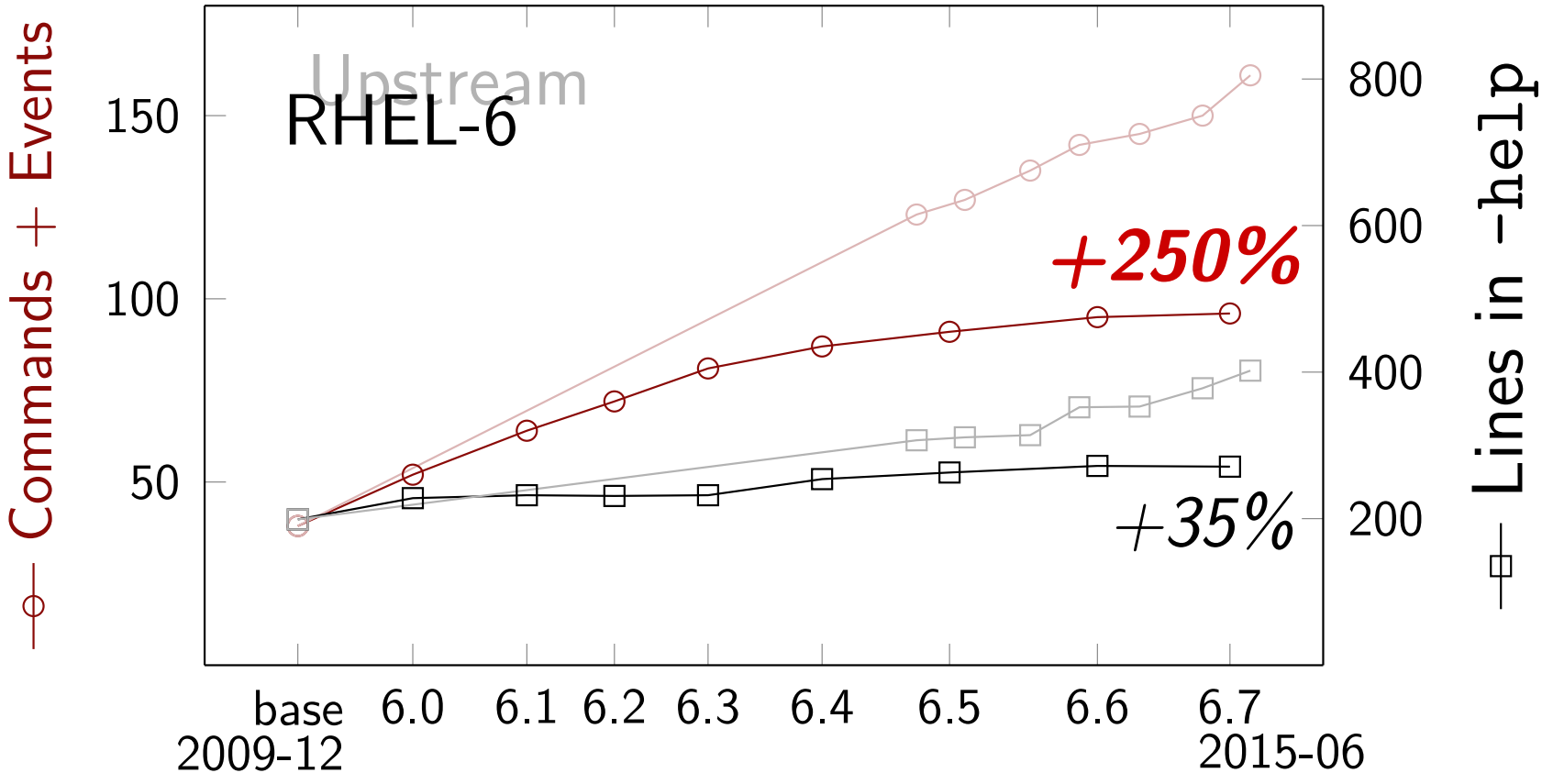
- Tie to a specific build of QEMU
  Wrong talk, you can break for coffee now
- Figure out what the QEMU it got can do
  Easiest when interfaces support introspection

Trouble is our interfaces don't fully support it, yet

Part II
Prior work

Version numbers?

QEMU says 0.12.1, but...



Backports make the upstream version meaningless

Downstream version?

We provide for downstreams adding their version

But: not always stepped
$\rightsquigarrow$ Downstream version often meaningless, too

Even if it wasn't:
#downstreams $\times$ #releases $=$ heaps of versions

Upstream wise guy shrugs *"downstream problem"*

Version numbers insufficient!

Upstream 2.4 development cycle:

```
$ git-diff --shortstat v2.3.0..v2.4.0
 1414 files changed, 72635 insertions(+),
 25875 deletions(-)
$ git-log --oneline v2.3.0..v2.4.0 | wc -l
2147
```

Version for 1901 out of 2147 commits: 2.3.50

Development can't wait for the version to change!

Upstream wise guy gets enlightened

Just try to use it

Workable in simple cases

Example: libvirt tries QMP `inject-nmi`,
falls back to old HMP `nmi`

Complex, slow, fragile in not so simple cases

- Recovery may need to recognize exact error
- Probe by trying may need complex scaffolding
- Probe must avoid unwanted effects

A real-world failure of “just try”

`block-commit` new in v1.3, and libvirt just tried it:

- Run `block-commit`, and if it succeeds
- wait for event `BLOCK_JOB_COMPLETED`

Before v2.0, `block-commit` fails for active layer
Since then, it succeeds, but requires manual
`block-job-complete` to complete

$\rightsquigarrow$ Old libvirt *hangs* on `BLOCK_JOB_COMPLETED`

Relying on behaviour in error cases is fragile

Pretend to be human: read help

Examples: `-help`
`-device help`
`-device virtio-net,help`
`-drive format=help`

Drawbacks:

- Parsing help is painful and fragile
- Help text becomes de facto ABI
- Help is incomplete (e.g. no `-drive if=help`)

Everyone did this until QEMU grew real interfaces

QMP query-commands

Example:

```
QMP> { "execute":"query-commands" }
{ "return":[ ... { "name":"eject" }, ... ] }
```

This is *very limited* QMP interface introspection:

- can check presence of commands
- silent on arguments & results

Plenty useful, but we *need* arguments & results now

QMP query-command-line-options

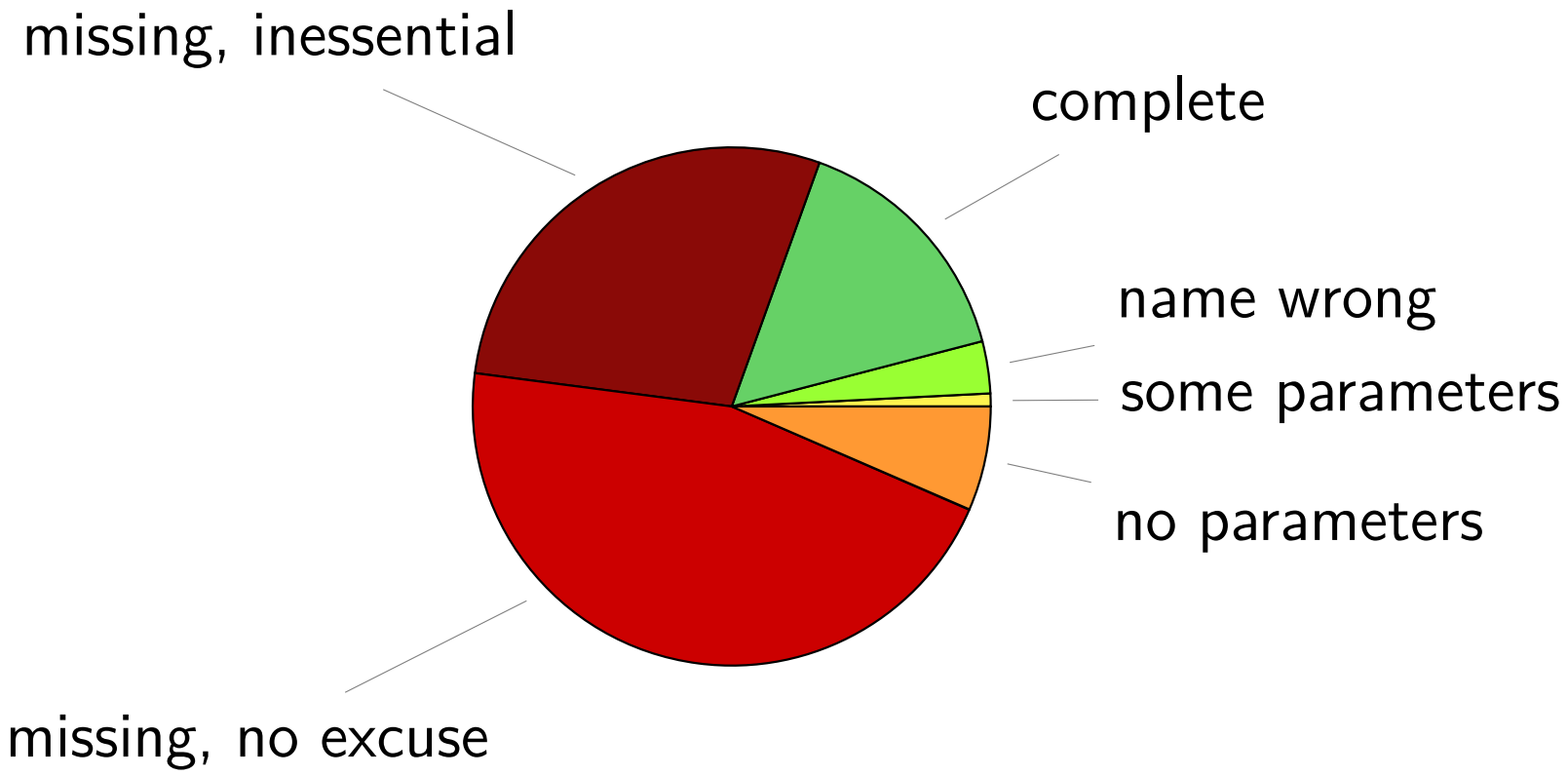
Example:

```
QMP> { "execute":"query-command-line-options" }
{ "return":[ ...
{ "option":"memory", "parameters":[
{ "name":"slots", "type":"number" },
{ "name":"size", "type":"size" } ],
... } ] }
```

This *tries* to be command line introspection

What's wrong with it? Dear God, where to begin!

query-c-l-options is incomplete



Probably better than nothing

Certainly less than needed

query-c-l-options is inexpressive

Things we'd like to know, but it can't tell:

- Formats supported by `-drive`?
  It only tells us parameter `format` is "string"
- Parameters supported with `-chardev socket`?
  It tells us parameters supported by *any* backend

Structural weaknesses

Worse than merely incomplete

Look for a witness

Example: how libvirt probes for `-spice`

- New in v0.14
- No `query-command-line-options` back then
- With `-spice`, we added QMP `query-spice`
- Can probe that one with `query-commands`
- Use `query-spice` as a *witness* for `-spice`

Useful when direct probe is impractical
and a suitable witness exists

Add an ad hoc query command

Got several already
How many more?

Add an ad hoc query command

Got several already

How many more?

Can we get a joker instead?

Where do we stand now?

Current introspection solutions work,
but won't cut it much longer:

- `query-command-line-options`
  too incomplete and inexpressive
- `query-commands` too limited
  we need arguments & results
- Adding a `query-FOO` for every FOO
  will result in a mess

Time to crack QMP introspection for real!

Part III
QMP Introspection

The basic idea

Interface introspection turns interface into data
QMP is defined by QAPI schema
Schema is *data*, so let clients query for it!

However:

- Geared towards humans, not machines
- Mixes up ABI aspects and internal detail

Instead of simply dumping QAPI schema:

- Expose only its QMP wire ABI aspects
- Design query output for machines

Cooking: query-schema

Coming in v2.5:

```
QMP> { "execute":"query-schema" }
{"return":[ ... {"name": "eject", ... ] }
```

Exposes QMP wire ABI as defined in the schema:

- Commands, events with arguments & results
- Arguments & results fully typed
- Enumerations (find supported values)
- Variants records
- ...

By yours truly, with heaps of help from Eric Blake

Let's introspect a command

QAPI Schema for `query-block`:

```
{ 'command': 'query-block',
  'returns': ['BlockInfo'] }
```

Relevant part of `query-schema`'s return:

```
{ "name":"query-block",
  "meta-type":"command",
  "arg-type":"15", "ret-type":"101" },
{ "name":"15", "meta-type":"object",
  "members":[] },
{ "name":"101", "meta-type":"array",
  "element-type":"183" }
```

Let's introspect a command

QAPI Schema for `query-block`:

```
{ 'command': 'query-block',
  'returns': ['BlockInfo'] }
```

Relevant part of `query-schema`'s return:

```
{ "name":"query-block",
  "meta-type":"command",
  "arg-type":"15", "ret-type":"101" },
{ "name":"15", "meta-type":"object",
  "members":[] },
{ "name":"101", "meta-type":"array",
  "element-type":"183" }
```

"query-block" is a command

Let's introspect a command

QAPI Schema for `query-block`:

```
{ 'command': 'query-block',     # no arguments
  'returns': ['BlockInfo'] }
```

Relevant part of `query-schema`'s return:

```
{ "name":"query-bloc
  "meta-type":"comma
  "arg-type":"15", "
{ "name":"15", "meta-type":"object",
  "members":[] },
{ "name":"101", "meta-type":"array",
  "element-type":"183" }
```

It has an empty argument type
(normalized from no arguments)

Let's introspect a command

QAPI Schema for `query-block`:

```
{ 'command': 'query-block',
  'returns': ['BlockInfo'] }
```

Relevant part of `query-schema`'s return:

```
{ "name":"query-block",
  "meta-type":"command",
  "arg-type":"15", "ret-type":"101" },
{ "name":"15", "met
  "members":[] },
{ "name":"101", "meta-type":"array",
  "element-type":"183" }
```

Return type is array of BlockInfo

Let's introspect a command

QAPI Schema for `query-block`:

```
{ 'command': 'query-block',
  'returns': ['BlockInfo'] }
```

Relevant part of `query-schema`'s return:

```
{ "name":"query-block",
  "meta-type":"command",
  "arg-type":"15", "ret-type":"101" },
{ "name":"15", "meta-type":"object",
  "members":[] },
{ "name":"101", "meta-type":"ar
  "element-type":"183" }
```

Type name 'BlockInfo' masked (not ABI)

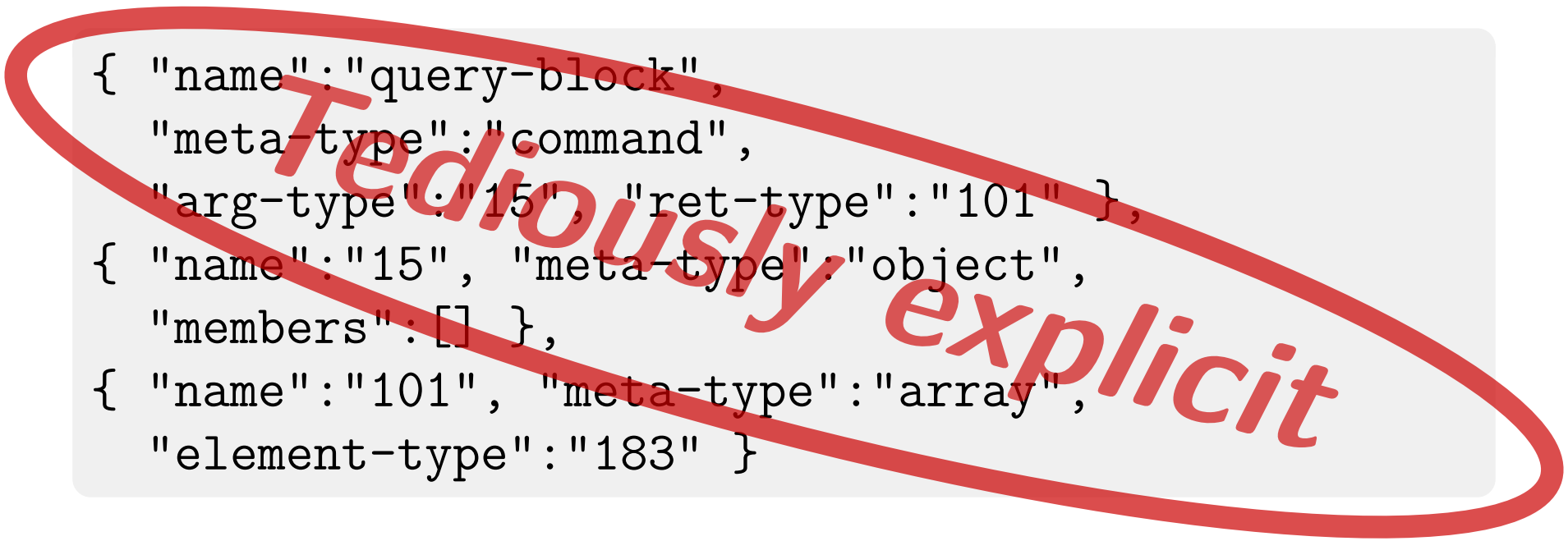
Let's introspect a command

QAPI Schema for `query-block`:

```
{ 'command': 'query-block',
  'returns': ['BlockInfo'] }
```

Relevant part of `query-schema`'s return:

```
{ "name":"query-block",
  "meta-type":"command",
  "arg-type":"15", "ret-type":"101" },
{ "name":"15", "meta-type":"object",
  "members":[] },
{ "name":"101", "meta-type":"array",
  "element-type":"183" }
```



Let's introspect introspection!

QAPI Schema for `query-schema`:

```
{ 'command': 'query-schema',
  'returns': ['SchemaInfo'] }
```

Like query-block, with SchemaInfo instead of BlockInfo

Relevant part of `query-schema`'s return:

```
{ "name":"query-schema",
  "meta-type":"command",
  "arg-type":"15", "ret-type":"129" },
{ "name":"15", "meta-type":"object",
  "members":[] },
{ "name":"129", "meta-type":"array",
  "element-type":"203" }
```

Drill down into SchemaInfo

```
{ 'union': 'SchemaInfo',
  'base': 'SchemaInfoBase',
  'discriminator': 'meta-type',
  'data': {
      'command': 'SchemaInfoCommand',
      'enum': 'SchemaInfoEnum',
      'object': 'SchemaInfoObject',
      ... } }
{ 'struct': 'SchemaInfoBase',
  'data': { 'name': 'str',
  'meta-type': 'SchemaMetaType' } }
{ 'enum': 'SchemaMetaType',
  'data': ['builtin', 'enum', 'array', 'object',
           'alternate', 'command', 'event'] }
```

Drill down into SchemaInfo

```
{ 'union': 'SchemaInfo',
  'bas
  'dis
  'dat

{ 'str
  'dat
  'met
{ 'enu
  'dat
```

QAPI schema gobbledygook decoded:

- `SchemaInfo` is a *variant record*
- Common members: `name`, `meta-type`
- Variant members depend on meta-type:

meta-type	variant members
command	`arg-type`, `ret-type`
array	`element-type`
object	`members`, `tag`, `variants`
...	

Introspect SchemaInfo

```
{ "name":"203", "meta-type":"object",
  "members":
      [{ "name":"name",      "type":"str" },
       { "name":"meta-type", "type":"271" }],
  "tag":"meta-type",
  "variants":
      [{ "case":"builtin",   "type":"272" },
       { "case":"enum",      "type":"273" },
       { "case":"array",     "type":"274" },
       { "case":"object",    "type":"275" },
       { "case":"alternate", "type":"276" },
       { "case":"command",   "type":"277" },
       { "case":"event",     "type":"278" }] }
```

Introspect SchemaInfo

```
{ "name":"203", "meta-type":"object",
  "members":
      [{ "name":"name",       "type
       { "name":"meta-type", "type":"271" }],
  "tag":"meta-type",
  "variants":
      [{ "case":"builtin",   "type":"272" },
       { "case":"enum",      "type":"273" },
       { "case":"array",     "type":"274" },
       { "case":"object",    "type":"275" },
       { "case":"alternate", "type":"276" },
       { "case":"command",   "type":"277" },
       { "case":"event",     "type":"278" }] }
```

It's an object type

redhat

Introspect SchemaInfo

```
{ "name":"203", "meta-type":"object",
  "members":
      [{ "name":"name",      "type":"str" },
       { "name":"meta-type", "type":"271" }],
  "tag":"meta
  "variants"
      [{ "case":"builtin",   "type":"272" },
       { "case":"enum",      "type":"273" },
       { "case":"array",     "type":"274" },
       { "case":"object",    "type":"275" },
       { "case":"alternate", "type":"276" },
       { "case":"command",   "type":"277" },
       { "case":"event",     "type":"278" }] }
```

It has a name and a meta-type (always)

Introspect SchemaInfo

```
{ "name":"203", "meta-type":"object",
  "members":
      [{ "name":"name",      "type":"str" },
       { "name":"meta-type", "type":"271" }],
  "tag":"meta-type",
  "variants":
      [{ "case":"builti
       { "case":"enum",
       { "case":"array",     "type":"274" },
       { "case":"object",    "type":"275" },
       { "case":"alternate", "type":"276" },
       { "case":"command",   "type":"277" },
       { "case":"event",     "type":"278" }] }
```

It has variants,
and this is their tag

Introspect SchemaInfo

```
{ "name":"203", "meta-type":"object",
  "members":
      [{ "name":"name",      "type":"str" },
       { "name":"meta-type", "type":"271" }],
  "tag":"meta-type",
  "variants":
      [{ "case":"builtin",
       { "case":"enum",
       { "case":"array",     "type":"274" },
       { "case":"object",    "type":"275" },
       { "case":"alternate", "type":"276" },
       { "case":"command",   "type":"277" },
       { "case":"event",     "type":"278" }] }
```

Each case applies to a value of the tag's type

Introspect SchemaInfo

```
{ "name":"203", "meta-type":"object",
  "members":
      [{ "name":"name",       "type":"str" },
       { "name":"meta-type", "type":"271" }],
  "tag":"meta-type",
  "variants":
      [{ "case":"builtin",    "type":"272" },
       { "case":"enum",       "type":"273" },
                              "type":"274" },
                              "type":"275" },
                    ate",     "type":"276" },
       { "case":"command",    "type":"277" },
       { "case":"event",      "type":"278" }] }
```

Variant members are just another object type

The types of its name & tag

The name's type is "str"

```
{ "name":"str",
  "meta-type":"builtin",
  "json-type":"string" }
```

It's a built-in type

and on the wire it's JSON string

```
{ "name":"271", "meta-type":"enum",
  "values":
      ["builtin", "enum", "array", "object",
       "alternate", "command", "event"] }
```

The types of its name & tag

```
{ "name":"str",
  "meta-type":"builtin",
  "json-type":"string" }
```

Tag's type is an enumeration

```
{ "name":"271", "meta-type":"enum",
  "values":
      ["builtin", "enum", "array", "object",
       "alternate", "command", "event"] }
```

with these members

SchemaInfo's enum variant

The case's type is an object type

```
{ "name":"273",
  "meta-type":"object",
  "members":
       [{ "name":"values", "type":"270" }] },
```

It has just one member "values"

```
{ "name":"270", "meta-type":"array",
  "element-type":"str" }
```

SchemaInfo's enum variant

```
{ "name":"273",
  "meta-type":"object",
  "members":
       [{ "name":"values", "type":"270" }] },
```

The type of "values" is array of str

```
{ "name":"270", "meta-type":"array",
  "element-type":"str" }
```

Quick peek under the hood

- QAPI schema is compile time static so far
- SchemaInfo is generated from it
- Generator is 160 SLOC of Python

  The necessary refactoring of core, however...
- Complete info is a bit over 70KiB
- Should probably support caching it

  (put hash in QMP greeting)
- Still work in progress

  `http://repo.or.cz/qemu/armbru.git`

  qapi-introspect

Part IV
Future work

QMP introspection limitations

Known issues:

- Can see only qapified commands
  ⇝ Can't see `device_add`
- Can see only qapified arguments & results
  We cheat for `netdev_add`...
  ⇝ Can't see most of `netdev_add`'s arguments
- Only as good as the qapification
  `add_client` takes arguments `protocol`, `tls`
  `tls` accepted only when protocol supports it
  ⇝ Can't see which protocols support TLS

Clean up qapification of netdev_add

Need to

- qapify its type-specific arguments. . .
- without upsetting the QMP wire format

Wire format matches QAPI/QMP's flat union type
Possible solution:

- Support unions as command arguments
- Code up the matching flat union type

Qapify device_add

Wire format like `netdev_add`:

common + driver-specific arguments

But: drivers collected only at run time!

QAPI schema fixed at compile time...

Choices:

- Collect drivers at compile time? *Hard...*
- Make QAPI schema dynamic? *Hard...*
- Forgo driver-specific arguments in schema? *Defeats introspection...*

No conclusion, yet

QAPI follow-up work

On the way to introspection, we

- got ourselves real test coverage
- replaced our internal schema representation
- fixed many bugs, and marked more as FIXME
- plugged many documentation holes

Left to do:

- Fix the FIXMEs
- Finish transition to new internal representation
- Clean up schema language and generated code
- Review schema for imprecise qapifications

What about command line?

Introspection needs the interface as data
Good: our command line definition is data
Bad: not QAPI, less expressive, leaves more to code

Choices:

- Build non-QAPI command line introspection?
  Only as good as the data. . .
- Rebase command line onto QAPI? *Hard. . .*
- Use QMP introspection as witness instead?
  Assumes suitable witness exists. . .

No conclusion, yet

Questions?