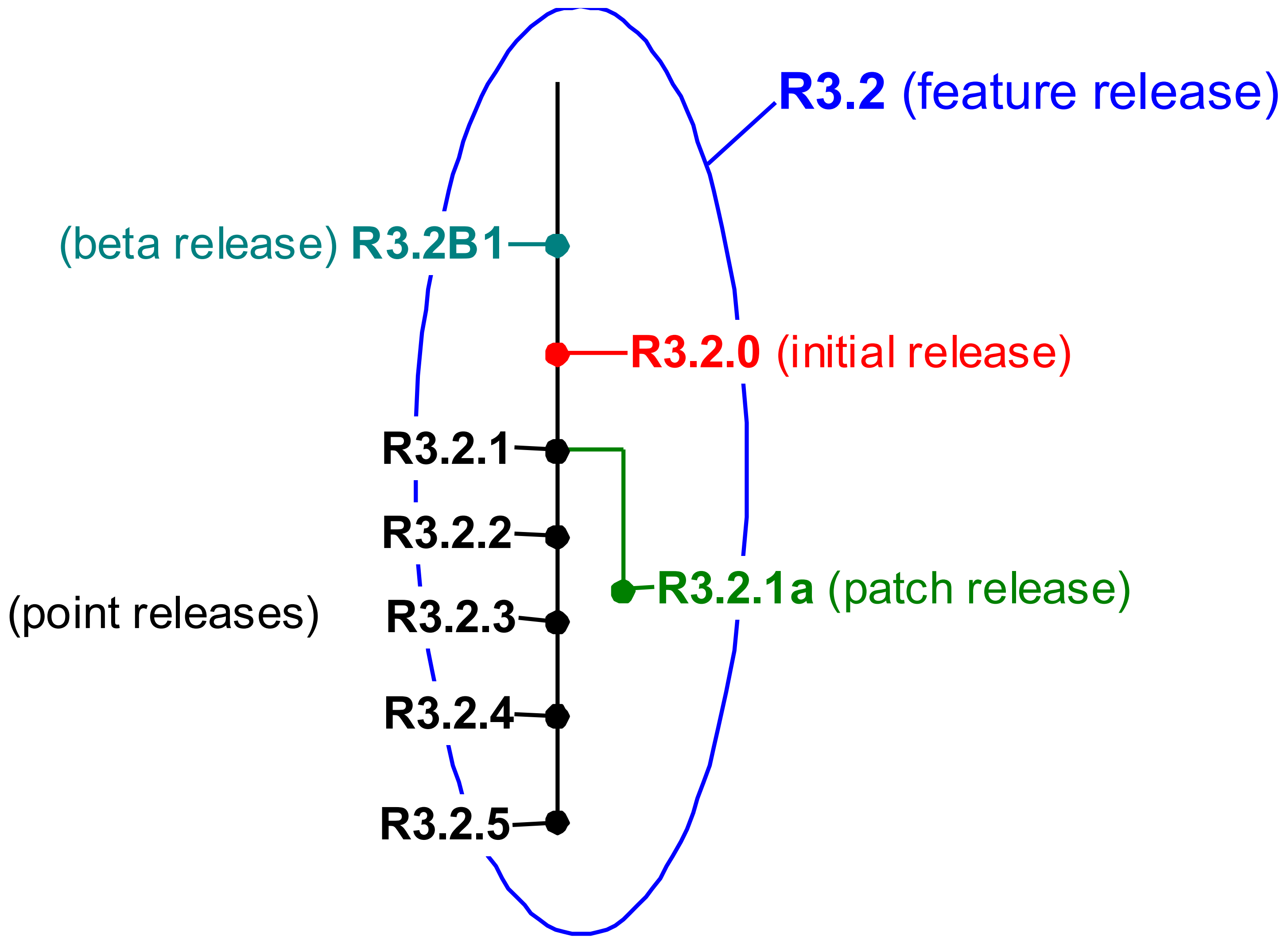
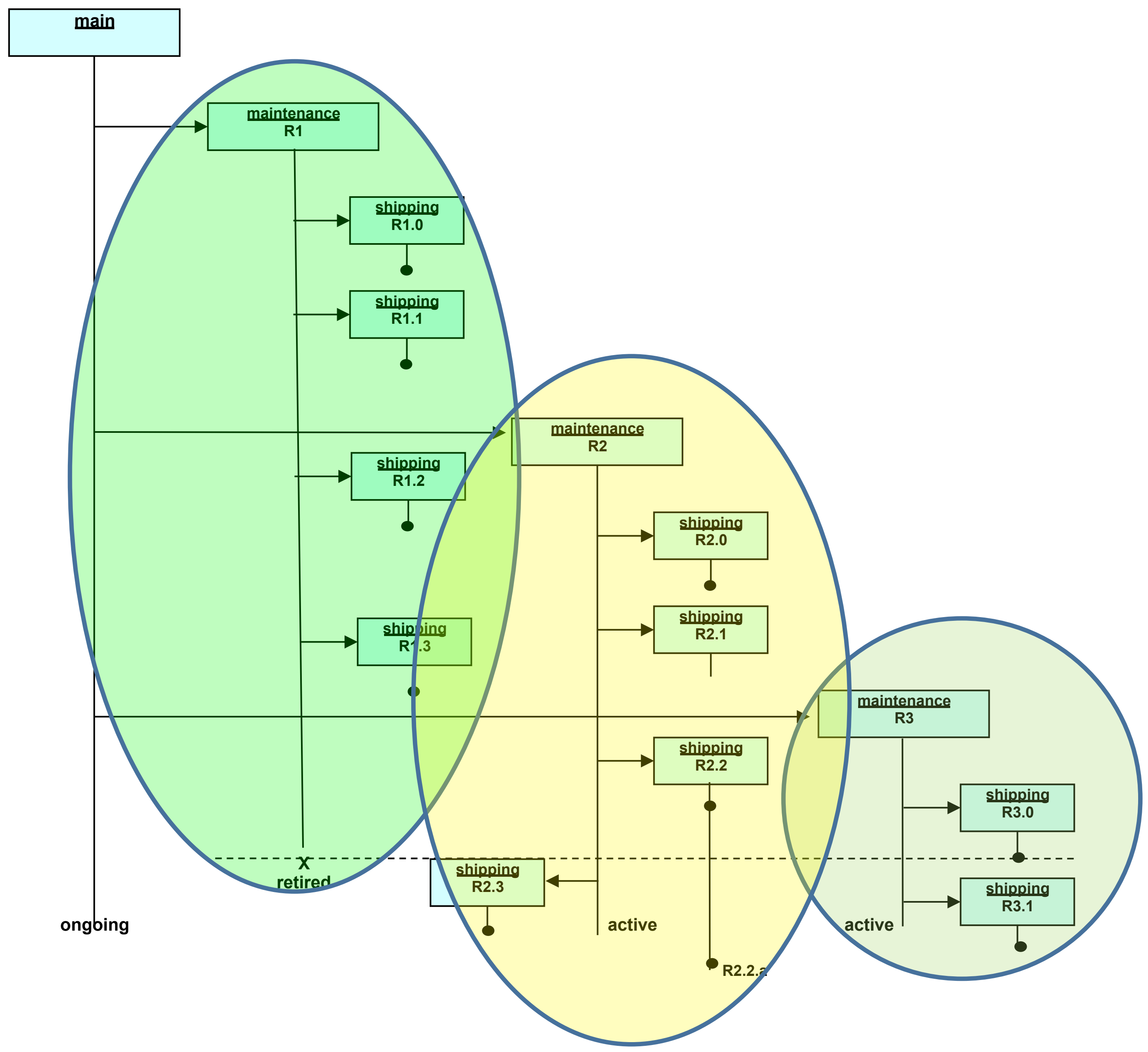
releases

concepts & terminology



another example



cost of feature releases

- considerable overhead associated with a feature release
  - system testing
  - marketing collateral
  - launch events
  - customer/partner briefings
  - new training courses & material
  - new training internally
  - burn CDs and shrink-wrap boxes
  - website downtime
  - …

cost of feature releases (2)

- largest cost of them all
  - – increased maintenance burden from supporting another version in the field
    - reproduce bugs in multiple codelines
    - decide what/when to fix
    - re-test, re-release
- maintenance releases are much less costly
  - – regression tests will catch problems

simultaneous release support

- generally support 2 feature release maintenance streams
- sometimes need to support 3 or more!
- MUST try to limit this
- if not, maintenance will erode and company will not be able to respond quickly to market conditions
  - extreme is separate release per customer
- how do web apps and mobile apply?

simultaneous release support (2)

- opportunity cost of developers
  - a trained developer is a scarce and valuable resource!
  - new features or maintenance tradeoff
  - opportunity cost of maintenance is the revenue the new feature might have brought in
  - opportunity cost of feature development is customer loss due to lack of maintenance

time between releases

- feature releases are costly:
  - therefore increase the time between releases
- but, customers want more features
  - therefore decrease the time between releases
- but, they also want stability in their own IT environment
  - therefore increase the time between releases
  - sometimes customers get very sticky on old releases
  - need to make the new release compelling to end-users

time between releases (2)

- what if one customer or prospect wants a new feature?
  - new feature release?
  - probably not

- what if the market condition changes rapidly?
  - cut short current release to rush it out?
  - go back to last release, extend it, and put that out?
  - costly: because of short release cycle will need to support > 3 releases in the field.

pushing back

- a successful development manager will need to distinguish between people asking for things that can be pushed off, and truly urgent things
  – everything is presented as the latter!

- track the request back to its source, personally
  – will learn the true nature of thee request
  – can deal with 80% of "urgent" requests in this manner

features in maintenance releases

- tried pushing back
- cannot justify a new feature release
- customer/prospect still wants/needs features earlier than the next scheduled feature release
- what now?

- slip new features into a maintenance release
- in theory, maintenance releases should change no externally visible program behavior (other than to correct it if faulty)
- what the heck, do it anyways
- does not have the cost of a new feature release
- why not?

features in maintenance releases (2)

- cannot introduce new code without introducing new defects
- reasons for adding code: feature, bug fix

- if fixing bugs:
  – fix 2, add 1: trend is good: -1
  – will eventually get them all – converge on quality
- if also doing new features:
  – fix 2, add 1, add new feature, add 4: trend is bad: +3 – diverging quality

negative leveraging

- the new feature is only useful to one customer
- the defects introduced as a result can negatively impact every customer
- because touching code risks breaking ANYTHING, ANYWHERE
- customers get irate if a "maintenance release" breaks previously working functionality
  - danger even when just fixing defects
  - gets much worse if adding features

release proliferation

- if your software is generally of poor quality customers will be slow to upgrade due to fear of more bugs
  - leads to supporting many releases

- EVEN WORSE: if customers come to fear maintenance releases the situation multiplies
  - customers may insist on patches to their maintenance level
  - turns every point release into its own maintenance stream!

mitigating the consequences

- "can we do it between releases?"
- "it's a web app, so it's easy"

- ugh! if absolutely forced to, then:
  - MUST have excellent regression testing environment
  - segregate new functionality with runtime configuration switch
    - code review to ensure switch off == no new code in the system
  - try not to allow this to set a precedent

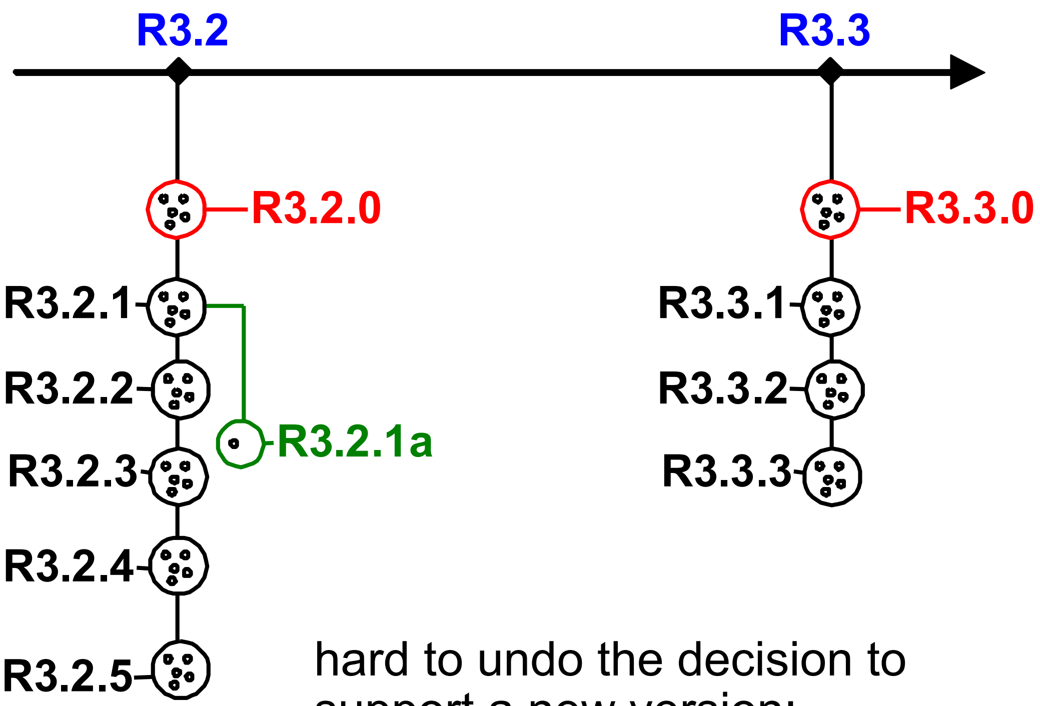
versions

versions

- as distinguished from "releases".
- different variants of the same software
  - differ in small ways
- does not apply as much to SaaS

must support:

- stream of maintenance releases for each version
- each feature release will continue to ship that version
  - ideally at the same time



version reasons:

- multiple hardware platforms
- multiple os's
- multiple databases
- multiple app frameworks
- multiple partner software
- security
- functional tiers
- demoware
- translations
- customizations

hard to undo the decision to support a new version:

- some customer now relies on it

cost of versions

- surprisingly costly to support many versions
  - not the development cost: relatively cheap – just another feature
  - ongoing maintenance costs
- technical means:
  - different code (linked differently or `#ifdef`'d)
  - run-time switches (e.g., dynamically detect version of Windows and change API calls appropriately).
  - different dev platform and tools
  - binary-compatible: different test environments
- in any case:
  - testers must test all supported versions
  - coders must bear in mind they are supporting multiple versions
  - must track down bugs in each version and fix

javascript web apps

version proliferation

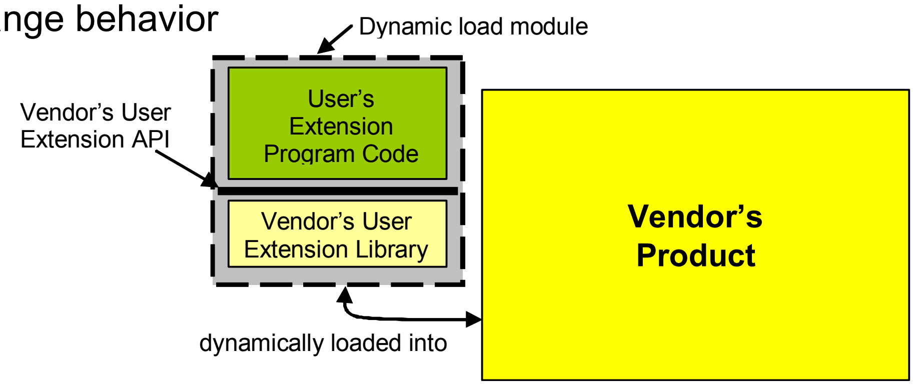
- software company will support many versions in hopes sales will increase
  - each version opens up a new market segment
- danger: too hastily commit to supporting too many versions
- be aware of costs and push-back
- if in the business of supporting many versions:
  - architect the software well to support it
  - construct a superb multi-platform automated build/test environment

customized software

- a different variant of the software for important customers
  - static methods: require a distinct executable
  - dynamic methods: same executable
    - run-time switches
    - alternate dll's (.dylib, .so)
- if customization required on feature release boundary
  - evaluate if feature's dev opportunity costs are worth the revenue
- if customization required sooner
  - either:
    - carefully insert changes into the point release stream
    - `#ifdef` all code and build a unique executable for the customer
  - can we merge the changes into the next feature release?
  - nothing very palatable here
- better to build in enough configurability that customers do not require customizations
  - GUI-based configuration
  - scripting-based configuration

user extension API

- allows customers to implement their own features into the software
- nowadays called "SOA" – implemented using web methods
- danger:
  - must support the API forever more
  - even if one already exists internally:
    - clean it up
    - identify public versus private APIs
    - document it
    - train customers on it
    - hire programmers to provide help desk support on it
      - support becomes "debug the customer's code"
    - maintain it unchanged
      - do not inadvertently change behavior
    - market it and sell it
    - consult on it
      - write it once
      - support it forever?
        - » paid/unpaid?


Dynamic load module
Vendor's User Extension API
User's Extension Program Code
Vendor's User Extension Library
Vendor's Product
dynamically loaded into