We Can Have Nice Things

Neovim and the state of text editor art in 2019

Neovim  https://neovim.io/
VimConf 2019  https://vimconf.org
Presenter

- Justin M. Keyes  https://sink.io/
- Nvim maintainer.
  - Roadmap, Vision, Docs
  - Release-management
  - Decision-fatigue
- I'm nobody. Feature, not a bug.
  - No celebrity-point-of-failure.

Previous talks

- 2016:  https://youtu.be/9Yf3dJSYdEA
- 2017:  https://youtu.be/wQh7saOHE5g
- Part 1: State of the art
- Part 2: Neovim tech
State of the art

- Joe Armstrong: more software = more entropy¹
- Rich Hickey: simple is not easy²
- Gary Bernhardt: Destroy All Software
- Alan Kay: "Computers are beautiful. But we have a know-nothing culture trying to use them."
- Jonathan Blow: revisit software foundations/history³

1: "The Mess We’re In" [https://www.youtube.com/watch?v=lKXe3HUG2I4](https://www.youtube.com/watch?v=lKXe3HUG2I4)
2: [https://www.infoq.com/presentations/Simple-Made-Easy/](https://www.infoq.com/presentations/Simple-Made-Easy/)
3: [https://www.youtube.com/watch?v=pW-SOdj4Kkk](https://www.youtube.com/watch?v=pW-SOdj4Kkk)
Neovim goals

- New text editor that doesn't throw away Vim.
- Extensible Vim
- Ubiquitous Vim
- Hackable Vim
- Push Vim into new territory

Goal is not to replace Vim, goal is More Vim.

- Target a subset of Vim's audience.
  - Vim targets "every conceivable user".
  - Nvim audience is "people who want more potential + less entropy". YMMV.
Themes

● System vs Application
● Legacy paradox
● Leverage = (impact / cost)
System vs Application

- Roles depend on context
  - Example: producer vs consumer
- Humans are software (flexible), not hardware
- Inflexible software = hardware
- System ~ architecture (hard to change)
- Ad-hoc is a valuable use-case
Legacy paradox

- Burden: support existing dependants
- Benefit: start from 9000 instead of 0
Leverage

Leverage = (impact / cost)
- impact = total effect (usage x time)
- cost = effort, human-hours, maintenance burden, ...

Low leverage: shallow features (increased entropy)
High leverage: deep extensibility
THE FUTURE OF TEXT EDITING
THE FUTURE OF TEXT EDITING

... is the past
The future of text editing!

IDE projects have huge teams for marketing, development.

- Q: How is it that Vim/Emacs are still relevant, and even outlast once-popular products like Eclipse, Netbeans, Textmate, Sublime?
- A: IDEs serve the common case (mainstream). Vim/Emacs focus on a niche. Mainstream ignores the niche.
The future of text editing!

How to create a plug-in:
Vimscript: plugin/foo.vim
Lua: lua/foo.lua

'runtimepath' works like $PATH, $PYTHON_PATH, Java classpath.
Easy to create and share plugins.
The future of text editing!

IDE projects are building sophisticated analysis and refactoring tools.

Neovim targets "server" and "client" roles equally.

Hosted = parasite = good design :)

(neovim logo)
Legacy

"Windows Phone was actually an amazing platform for both users and developers, and shows a fundamental rule of technology: There Is No Third Ecosystem."
- former Nokia employee

IOW: ecosystems tend to be winner-takes all (80% of users will use the top few, the rest is "long tail")

cf. textmate grammars, javascript, Vim plugins, ...

https://news.ycombinator.com/item?id=16370602
Worse is better

"It is often undesirable to go for the Right Thing first."

- Ship *half* of the Right Thing so that it spreads like a virus.
- *Then* take the time to improve it to 90% of the Right Thing.


genetic model:
- IDEs, other random text editor projects => side effect: LSP, semantic code nav
- bitcoin: mining => blocks
worse is better: TCP/IP, plain text, Javascript, Vim, Emacs, C, Von Neumann, ...
Worse is better
Worse is better

Vim's missing 50%:
● Imperfect design => bad perf: macros, long lines, syntax
● Vimscript is slow: no AST, ad-hoc impl
● :vimgrep is slow, :syntax is slow, ...
● Legacy arch: 600+ globals, high coupling, TUI assumption
● Inconsistent UI/behavior: win_getid() vs getwininfo()

inconsistent UX:
- :filter doesn't work with every command, because command impls are ad-hoc
- why is 'statusline' a DSL instead of a function?
- 'fooexpr' vs 'fooprg' vs 'foofunc' options
- function() vs funcref() vs Funcref
- :terminal buffers should work like any other buffer/channel
- v:none and v:null
Vim: the good parts

What do we like about Vim?

● Powerful (do a lot, with a little) (AKA: leverage) => multiply capabilities (new techniques, compose actions, ...)
● Usable (:help, completion, quickfix, swapfiles, ...)
● Portable (easy to get, cross-platform)
● Fast/small (actually a subset of "portable")
● Flexible (easy to create plugins, change behavior)
Why fork Vim?

Better question: why start from scratch?

Text editing is hard\(^1\): multibyte rendering, layout, cursor positioning, line-wrapping

Vim iceberg: shell handling, encoding, completion, Vim regex, quickfix ... Massive plugin archive.

Focus on usability and extensibility => remove anti-features, dead-ends.

Dead-ends are costly for usability.

\(^1\): [https://lord.io/blog/2019/text-editing-hates-you-too/](https://lord.io/blog/2019/text-editing-hates-you-too/)
Why fork Vim?

Repair is as important as innovation

*Maintenance lacks the glamour of innovation. It is mostly noticed in its absence—the tear in a shirt, the mould on a ceiling, the spluttering of an engine.*

IOW: legacy is important.
Vim way
# Vim way

:helpgrep [Vv]im way

<table>
<thead>
<tr>
<th>Examples</th>
<th>Vim way</th>
<th>Vi-compatible way</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;uu&quot;</td>
<td>two times undo</td>
<td>no-op</td>
</tr>
<tr>
<td>&quot;u CTRL-R&quot;</td>
<td>no-op</td>
<td>two times undo</td>
</tr>
</tbody>
</table>
Vim way

Vim way, IMO:

- Macro-friendly: "Vim is optimized for repetition."\(^1\)
- Common conventions (re-use concepts)
- Optimize ad-hoc: :nn instead of set_mapping()
- Leverage external tools
- DWIS not DWIM
- Keystroke-driven: gj instead of move_cursor()

\(^1\): [Practical Vim, 2nd Edition by Drew Neil](https://example.com)
Unix way


simple, short, clear, modular, and extensible code ... favors composability as opposed to monolithic design.

Vim way is unrelated to the Unix way.
Vim way

:help design-not

---
VIM IS... NOT *design-not*

-- Vim is not a shell or an Operating System. You will not be able to run a shell inside Vim or use it to control a debugger. This should work the other way around: Use Vim as a component from a shell or in an IDE.

+- Vim is not a shell or an Operating System. It does provide a terminal window, in which you can run a shell or debugger. E.g. to be able to do this over an ssh connection. But if you don't need a text editor with it is out of scope (use something like screen or tmux instead).
Vim way

:help shell-window

There have been questions for the possibility to execute a shell in a window inside Vim. The answer: you can't! Including this would add a lot of code to Vim, which is a good reason not to do this.

Vim is no longer afraid to a lots and lots of code: xdiff, libvterm, big plugins (netrw is 11k LoC), ...

http://vimdoc.sourceforge.net/htmldoc/tips.html#shell-window
Vim way

:help design-improved

There is no limit to the features that can be added. Selecting new features is based on (1) what users ask for, (2) how much effort it takes to implement and (3) someone actually implementing it.
Neovim way
Neovim way

- Usability
- Extensibility
Usability is high-leverage

When a small problem is fixed forever, the benefits accrete over time + users.

\[
\text{impact } \sim O(N \times M) \\
\text{cost } \sim O(1)
\]
Extensibility is high-leverage

Vim users already know this, that's why they like :make, 'formatprg', :, plugins, ...

Opposite of "kitchen sink".
# Extend Vim

<table>
<thead>
<tr>
<th>Nvim</th>
<th>Vim</th>
</tr>
</thead>
<tbody>
<tr>
<td>:terminal</td>
<td>:terminal</td>
</tr>
<tr>
<td>tarruda, others</td>
<td>Bram</td>
</tr>
<tr>
<td>buf-update</td>
<td>buf-update</td>
</tr>
<tr>
<td>phodge</td>
<td>Bram</td>
</tr>
<tr>
<td>docs</td>
<td>docs</td>
</tr>
<tr>
<td>justinmk</td>
<td>Bram</td>
</tr>
<tr>
<td>eval</td>
<td>eval</td>
</tr>
<tr>
<td>zyx</td>
<td>Bram</td>
</tr>
<tr>
<td>extmarks</td>
<td>textprop</td>
</tr>
<tr>
<td>timeyy</td>
<td>Bram</td>
</tr>
<tr>
<td>floatwin</td>
<td>popup</td>
</tr>
<tr>
<td>bfredl</td>
<td>Bram</td>
</tr>
<tr>
<td>job/chan</td>
<td>job/chan</td>
</tr>
<tr>
<td>tarruda, bfredl</td>
<td>Bram</td>
</tr>
<tr>
<td>cmake</td>
<td>?</td>
</tr>
<tr>
<td>inccommand</td>
<td>?</td>
</tr>
<tr>
<td>various</td>
<td>?</td>
</tr>
<tr>
<td>lua</td>
<td>?</td>
</tr>
<tr>
<td>zyx, bfredl, others</td>
<td>?</td>
</tr>
<tr>
<td>multiproc</td>
<td>?</td>
</tr>
<tr>
<td>abdelhakeem</td>
<td>?</td>
</tr>
<tr>
<td>paste</td>
<td>?</td>
</tr>
<tr>
<td>justinmk</td>
<td>?</td>
</tr>
</tbody>
</table>
Middle Ages 20XX - 2016
Middle Ages 20XX - 2016

- Text editor camp: "I don't need IDE features".
- IDE camp: "Text editing is not important".

Users must "choose a religion".
Middle Ages 20XX - 2016

Vim development
Middle Ages 20XX - 2016

- human-powered CI (Tony M. et al.)
- bad test coverage.
- Mailing-list-driven development.
Vim development is slow, it's quite stable and still there are plenty of bugs to fix. Adding a new feature always means new bugs, thus hardly any new features are going to be added now. I did add a few for Vim 7.3, and that did introduce quite a few new problems. Even though several people said the patch worked fine.

—Bram Moolenaar
10 Questions with Vim’s creator (2014):

Q: How can the community ensure that the Vim project succeeds for the foreseeable future?
A: Keep me alive.

Q: What does the future hold for Vim?
—Bram Moolenaar
Half-measures:
● FEAT_NETBEANS
● --remote (FEAT_CLIENTSERVER)
● ballooneval
● if_lua, if_python, if_tcl, if_perl, if_mzsch
● ...

select() is specified in POSIX.1-2001
event-loop: queue that dispatches event-handlers
Neovim vision
Neovim vision

https://neovim.io/charter/

- You shouldn't need to choose "editor" or "IDE".
- Can have both, by maximizing extensibility (Unix way).
Text editor heresy

Software treats censure as damage and routes around it. Inflexible=hardware (humans are software!)
Hardware (invariants) are valuable for building systems.
Ad-hoc tasks (exploration/applications) are antagonized by systems.
System = foundation
Application = edges/surface.

Vimscript, Ex commands, Vi are for ad-hoc tasks. Like a shell.
Text editor heresy

"Computers are beautiful. But we have a know-nothing culture trying to use them. It's like in the middle ages if you wanted to be a physicist you just had to get a pointed hat."
- Alan Kay
Text editor heresy

Use your OS to:
- Create a form? Build a UI? (widget library)
- Show a dialog?
- Display an image
- Orchestrate tasks (try `jobstart(...) ({callback})` in your shell!)
- Compose parts: VScode+Email=??
- Isolation/security (app/data sandbox)
- Play a sound

OS failed as a platform, because of "worse is better". Thus applications become platforms
The OS failed

Web browser = OS for GUI
- widgets
- scripting/plugins
- delivery
- sandboxing/isolation/security

See also Gary Bernhardt's The Birth & Death of JavaScript
The OS failed

Text editor = OS for TUI
● widgets
● scripting/plugins
● shell integration

todo :)  
● delivery ("app stores"?)
● sandboxing/isolation/security (Docker?)
Text editor heresy: `terminal`

```bash
$ ohcount nvim/src
Language          Files   Code
----------------  -----  ---------
c                238    212911 (2017: 174837)
vimscript        201    25907
lua               5     8500 (2017: 6461)
```

```bash
$ ohcount vim/src
Language          Files   Code
----------------  -----  ---------
c                236    348010 (2017: 317691)
vimscript        267    38480
```
Text editor heresy: :terminal

- :terminal is an elementary component (like buffer, pipe, pty). Not bloat.
- terminal.c is ~1k LOC.
- Vim screen.c:win_update() *function* is 1212 LOC.

Alan Kay: computers, not functions
Neovim effect
Neovim effect

Vim development: 2016-present
Neovim legacy

ed: line-addressable editing language
vi: normal-mode (AKA ex 2.0)
vim: +textobjects, +eval (Vimscript)
nvim: --embed, API, job-control, :terminal
Neovim status

No commits for 4 days. Is Neovim dead?
– anonymous user (2015)

P.S.: check https://github.com/neovim/neovim/pulse next time :)
Neovim status

Nvim

Contributors: 469
Commits: 14635 since 2014 (20% Vim patches)
2016: 6479 since 2014

Vim

Contributors: ? (300+)
Commits: 6565 since 2014
10729 since 2004
2016: 6553 since 2004

Does Nvim "divide" the Vim community?
Neovim status

- GitHub downloads: **310k+**
- Homebrew: **200k+** installs [https://brew.sh/analytics/install](https://brew.sh/analytics/install)
  - (2017: 100k)
- Reddit:
  - /r/neovim **11k** members
  - /r/vim 90k members
- Vibe: 30-50% of "Vim enthusiasts" (anecdotal/unscientific)
Neovim status

- **Hackable!**
  - 29 API clients (2017: 24)
  - 34 UIs (2017: 18)
- Easiest way to install "vim" on all major OSes: [https://github.com/neovim/neovim/releases](https://github.com/neovim/neovim/releases)
Neovim status

New API clients

- Dart client: https://github.com/smolck/dart-nvim-api
- Nim client: https://github.com/alaviss/nim.nvim
- Scala: https://github.com/viniarck/nvimhost-scala
- .NET: https://github.com/neovim/nvim.net
Inverse vandalism

34 UIs. 29 API clients. Why so many?

When "extravagance" becomes commodity, it yields new, useful technologies that previously seemed crazy.

Inverse vandalism: making things because we can.
- Alan Kay
Inverse vandalism

Vim depends on this phenomenon:
- Vim undotree is MVP (no compression/collapse)
- Vimscript parser/executor is 100% unoptimized
  - viable because of rapid hardware improvements
- Vim depends on filesystem cache (try --startuptime without it!)
Less is more

Rob Pike: "Less is exponentially more"¹

E.W. Dijkstra²

[PL/1 user] managed to ask for the addition of about fifty new “features”, little supposing that the main source of his problems could very well be that it contained already far too many “features”. The speaker displayed all the depressing symptoms of addiction ...

¹: https://commandcenter.blogspot.com/2012/06/less-is-exponentially-more.html
²: https://www.cs.utexas.edu/~EWD/transcriptions/EWD03xx/EWD340.html
Less is more

- Less "vim emulation" in IDEs.
- Less NIH: collaborate with third parties: libuv, libvterm, Lua, [treesitter](#), ...
  - Hard work. Reduces entropy.

"Feature" in statistics means "dimension": any differentiating characteristic. Entropy. Variation. This can be infinite.
Less is more: dead-ends

:help nvim-features-removed

● FEAT_XX
● t_xx
● test_xx()
● 'compatible' + 34 other options
● aliases: ex, exim, gex, gview, gvim, gvimdiff, rgview, rgvim, rview, rvim, view, vimdiff, eview, evim
● commands: :fixdel :open :tearoff
Less is more: docs

Lots of documentation in :help has been rewritten and often condensed.

Small but prominent examples:
nvim -h
man nvim
Less is more: CLI

The "-" file is implicit when sending text at startup. Equivalent:

```
echo foo | nvim -
echo foo | nvim
```

The "-s" arg takes "-" if you want the old behavior. Equivalent:

```
echo "ifoo" | nvim -s -
```

bonus: never pauses, never " Warning: Input is not from a terminal"
Less is more: composition

Nvim can be composed\(^1\) with other shell tools, the Unix way:

```
$ echo foo | nvim -Es +"%p" | tr o x
    fxx
```

\(^1\): https://sink.io/jmk/vim-social-life
Less is more: 'guicursor'

Configure cursor in TUI with 'guicursor' option.

    :set guicursor=n-v-c:block,i-ci-ve:ver25

t_xx is an anti-feature.
Neovim tech
Nvim 0.4/0.5 major topics

- API
- Decoupled UI
- Lua
Decoupled (externalized) UI

Decoupled:

- ext_popupmenu: completion menu
- ext_tabline: tab line
- ext_cmdline: command line
- ext_hlstate: highlight state
- ext_messages: messages
- ext_multigrid: windows, grids
- remote TUI

UI extension work tracking issue: [https://github.com/neovim/neovim/issues/9421](https://github.com/neovim/neovim/issues/9421)
Decoupled UI

Reminder: 34 UIs (2017: 18)
Why so many?
● It's easy/fun.
● Like the web: you don't have only 1 webapp. Potential for many apps: Firenvim, ActualVim.
● Not "Emacs". Not "kitchen-sink". This is the "unix way": extend, extend, extend, extend.
Decoupled UI

Structured protocol

[nvim] <-> [windows: win1, win2, ...]
[tabline: tab1, tab2, ...]
[cmdline]
[messages]
[popupmenu]
Decoupled UI

What does "structured" mean? Compare emacsclient…

terminal 1:
  emacs --daemon
  strace -o s.txt -s9999 -p $(pgrep emacs)
terminal 2:
  emacsclient -t
terminal 3:
  tail -F s.txt
Decoupled UI

*What does "structured" mean? Compare emacsclient...*

server opens client tty:

```
ioctl(7, TCGETS, {B38400 isig icanon...}) = 0
```

emacsclient loops over `recv()`.

server sends terminal sequences to draw statusline/minibuffer/etc:

```
write(7, "\33[10;1H\33[30m\33[47m-UIUU:@----F2
  \33[39;49m\33[1m\33[30m\33[47m*scratch* ... All (5,0)
(Lisp Interaction SP Undo-Tree ... \r\n", 812) = 812
```
Decoupled UI

… certainly [Xi editor is] inspired by Neovim.¹
—Raph Levien, author of Xi editor

¹: RustConf 2016 - A Modern Editor Built in Rust by Raph Levien
Decoupled UI: `ext_multigrid`

- Implements per-window grids
- Foundation for "multihead"
  - `ext_tabgrid` = multiple "screens" (like Emacs frames)
- Grids: popupmenu, messages, windows, screen
Decoupled UI: `ext_multigrid`

`:help ui-multigrid`

`["win_pos", grid, win, start_row, start_col, width, height]`
Decoupled UI: **ext_multigrid**

Per-window grids. Python REPL:

```python
>>> n.ui_attach(80, 10, rgb=False, override=True, ext_multigrid=True, ext_messages=True, ext_popupmenu=True)
```

```python
>>> while True: m=n.next_message(); print(m);
```
Decoupled UI: ext_multigrid

Per-window grids. Python REPL:

CTRL-W v

['notification', 'redraw',
  [['msg_showcmd', [[0, '^Wv']]], ['flush', []]]]

['notification', 'redraw',
  [['msg_showcmd', []],
  ['win_pos', [4, <Window(handle=1001)>, 0, 0, 40, 9],
   [2, <Window(handle=1000)>, 0, 41, 39, 9]],
   ^grid-id   ^win-id
  'tabline_update', [<Tabpage(handle=1)>, {'tab':
   <Tabpage(handle=1)>, 'name': '[No Name]'}]],
  ...
  ['grid_cursor_goto', [4, 0, 0], ['flush', []]]]
Decoupled UI: *ext_multigrid*

Per-window grids. Python REPL:

```
CTRL-W >
['notification', 'redraw',
 [ ['msg_showcmd', [[0, '^W>']]], ['flush', []]]]
['notification', 'redraw',
 [ ['msg_showcmd', []],
   ['win_pos', [4, <Window(handle=1001)>, 0, 0, 41, 9],
    [2, <Window(handle=1000)>, 0, 42, 38, 9]],
   ^grid-id  ^win-id

 ['tabline_update', [<Tabpage(handle=1)>, {'tab': <Tabpage(handle=1)>, 'name': '[No Name]'}]],
 ...
 ['grid_cursor_goto', [4, 0, 0], ['flush', []]]]
```
GUI: gonvim

https://github.com/akiyosi/gonvim
GUI: qnvim

Nvim embedded in Qt Creator IDE
https://github.com/sassanh/qnvim
by Sassan Haradji
GUI: veonim

:Veonim nc

TODO: alias to :smile

https://github.com/veonim/veonim
GUI: FVim: F# + Avalonia

- HiDPI support, "Nerd font"
- Low latency: 60FPS on 4K display
- To WSL Nvim: `fvim --wsl`
- To remote Nvim: `fvim --ssh user@host`
- Use custom Nvim: `fvim --nvim ~:/bin/nvim.appimage`
- Multi-grid <=> Multi-window mapping
- Extend with XAML -- UI widgets as Nvim plugins

https://github.com/yatli/fvim
GUI: FVim: smooth cursor pulse

```php
{ let buf, fin, m = startString args lexbuf
  if not skip then (STRING_TEXT (LexCont.TripleQuoteString(!args.ifdefStack,m))) else tripleQuoteString (buf, fin, m, args) skip lexbuf }

'$$'
{ fail args lexbuf (FSComp.SR.lexerReserved()) (WHITESPACE (LexCont.Token !args.ifdefStack)) }

'@'
{ let buf, fin, m = startString args lexbuf
  if not skip then (STRING_TEXT (LexCont.VerbamimString(!args.ifdefStack,m))) else verbatimString (buf, fin, m, args) skip lexbuf }

truewhite+
{ if skip then token args skip lexbuf
  else WHITESPACE (LexCont.Token !args.ifdefStack) }

offwhite+
{ if args.lightSyntaxStatus.Status then errorR(Error(FSComp.SR.lexerTabNotAllowed(), lexbuf.LexemeRange))
  if not skip then (WHITESPACE (LexCont.Token !args.ifdefStack)) else token args skip lexbuf }

"////" op_char*
{ // 4+ slash are 1-line comments, online 3 slash are XDoc
  let m = lexbuf.LexemeRange
  if not skip then (LINECOMMENT (LexCont.SingleLineComment(!args.ifdefStack,1,m))) else singleLineComment (None,1,m, args) skip lexbuf }

"\\\\" op_char*
{ // Match exactly 3 slash, 4+ slash caught by preceding rule
```
Vim: smooth cursor?

**patch 7.4.1890 GUI:** When channel data is received, cursor blinking is interrupted.

```
diff --git a/src/gui_gtk_x11.c b/src/gui_gtk_x11.c
index d497c7530c..601fafccd2 100644
--- a/src/gui_gtk_x11.c
+++ b/src/gui_gtk_x11.c
@@ -810,6 +810,12 @@
    #endif
    
    int
    +gui_mch_is_blinking(void)
    +{
    +    return blink_state != BLINK_NONE;
    +}
```

12 files changed, 40 insertions(+), 1 deletion(-)
GUI: Firenvim

ext_cmdline could be useful here...

@glacambre wow the reviewer doesn't know nvim and firenvim? Someone needs some general ed!

Thank you for the good work! Can't wait!

@glacambre commented 8 hours ago

Firenvim has been accepted on the Chrome Web store! You can find out how to install it here.
UI: from concept to PoC

With Neovim, UIs are plugins.

"Writing a GUI with Neovim is crazy easy. It took me about 4 hours, including learning a GPU framework."
- Ashkan Kiani

https://www.reddit.com/r/neovim/comments/dnb1vf/wip_cross_platform_gpu_accelerated_neovim/
More UIs

● GNvim: featureful/lightweight, built on Rust + GTK
  https://github.com/vhakulinen/gnvim
● VV: minimalist macOS Nvim GUI, WebGL-based text-rendering.
  https://github.com/vv-vim/vv
● Yours! UIs are plugins. Create a UI for your specific need or just for fun.
Decoupled UI: **remote TUI** (GSoC 2019)

$ nvim --listen server1  # PID 10219
$ nvim --connect server1  # PID 10221
$ pstree
tmux: server,13227
    ├─ bash,8738
    │   └─ nvim,10219 --listen server1
    │       └─ nvim,10220 --embed --listen server1
    └─ bash,9325
        └─ nvim,10221 --connect server1
Decoupled UI: remote TUI (GSoC 2019)

- Extensibility: Prepares Nvim as UI-RPC library, so GUIs and API clients are easier to implement.
- Reliability: Remove the TUI thread, TUI always runs as a coprocess.
- ext_tabgrid (WIP): different views of same server (multiplexing)
- Potential "alternative TUI": ext_cmdline?
- Not "replace tmux" (but sure, if you want)
API: multiproc (GSoC 2019)

- Multiproc = "job-control for Vimscript"
- GSoC project
- Author: Abdelhakeem Osama

Case study: asynchronous behavior for :vimgrep command family.

```
:vimgrep /buf_T/jg **/*.c **/*.h
:&vimgrep /buf_T/jg **/*.c **/*.h
```

executing: vimgrep /buf_T/jg **/*.c **/*.h
693 matches in 16.157729 seconds
executing: &:vimgrep /buf_T/jg **/*.c **/*.h
693 matches in 0.602333 seconds
speedup: 26.825251
API: nvim_api_get_context (GSoC 2019)

{'jumps': [{'file': 'man://select(2)', 'col': 129}, ...],
'vars': ['g:foo', 'val1', 'g:bar', 42],
'funcs': 'FugitiveExtractGitDir': {'sid': 48, 'source': 'function!
FugitiveExtractGitDir(path) abort
    let path = s:Slash(a:path)
    ...
endfunction'},
'opts': {
    'buf': {'binary': v:false, 'iskeyword': '@,48-57,_,192-255', ...}
    'global': {'winminheight': 1, 'inccommand': 'split', ... },
    'win': {'fillchars': 'msgsep: ''', ... },
    'regs': {'unnamed': v:true, 'name': '0', 'content': ['v[keys(v)[0]]']}
}
Nvim 0.4: wildoptions=pum, 'pumblend'

Popup wildmenu.

:set wildoptions=pum
:set pumblend=20

credit: Björn Linse
https://twitter.com/Neovim/status/1107014096908664832
Nvim 0.4: wildoptions=pum, 'pumblend'

Popup wildmenu.

:set wildoptions=pum
:set pumblend=20
Nvim 0.4: 'pumblend'

:set pumblend=40

credit: https://twitter.com/delphinus35
Nvim 0.4: floating windows

:help nvim_open_win()

- Show window at any (x,y) position.
  - pixel (sub-cell) offset for GUIs
- Useful for menus, selection UIs, dialogs
- No compromises: arbitrary control of **real windows + real buffers**.

*Running a terminal window in a popup seems like a total hack. No idea why anyone would want to do that.*

https://github.com/vim/vim/issues/4063#issuecomment-534228904
Nvim 0.4: floating windows

credit: ドッグ @Linda_pp  https://twitter.com/i/status/1103968541814874112
Nvim 0.4: floating windows

:let winblend = 30

credit: https://twitter.com/delphinus35/status/1144436863182049280
Nvim 0.4: floating windows

function! ColorWheel() abort
    const [center_x, center_y] = [&columns / 2.0, &lines
    const radius = min([&columns, &lines]) / 8.0 * 3
    ...
    while col < center_x + radius * s:pixel_ratio
        let row = center_y - radius
        while row < center_y + radius
            ...
            let winid = nvim_open_win(...)
            call nvim_win_set_option(winid, ...)
            ...
        endwhile
    endwhile
endfunction

credit: https://twitter.com/delphinus35/status/1144869405773295616
https://gist.github.com/delphinus/8b05cd9ad6e0f8f8e9be0d02b28f35df
Extensibility = leverage: Lua stdlib

Lua is designed for embedding. Lua is fast, LuaJit is *ridiculously* fast. Less is more: Lua language is super small, simple, complete (frozen).
Extensibility = leverage: Lua stdlib

Lua's lack of "batteries included" is a benefit. Nvim is the "stdlib".
Standard modules:
● inspect
● treesitter
● loop

Trivial to add new modules: put it on 'runtimepath'.
Extensibility = leverage: Lua stdlib

Future:
- init.lua (vimrc)
- More Lua, everywhere:
  - Implement (more) core features in Lua.
  - Lua REPL.
  - More standard modules (lpeg?)
  - More "ergonomics".
foo.vim:

let s:sum = 0
for i in range(1, 9999999)
  let s:sum = s:sum + i
endfor
call append('$', s:sum)

Time: 31.611 seconds
Vimscript vs Lua

```lua
foo.lua:
    sum = 0
    for i = 1, 9999999 do
        sum = sum + i
    end
    vim.api.nvim_call_function('append',
        {{'$', tostring(sum)}})
Time: 0.015 seconds
speedup: 31.611 / 0.015 = 2107 (two-thousand...)
```
Vimscript vs Lua

foo.vim:
    let s:sum = 0
    for i in range(1, 9999999)  " Parsed 10M times.
        let s:sum = s:sum + i  " Parsed 10M times.
    endfor                      " Parsed 10M times.
    call append('$$', s:sum)

ex_docmd.c:do_cmdline():
    ● copies command (script line), sends to ex_docmd.c:do_one_cmd()
    ● ex_docmd.c:do_one_cmd() recursively parses the line
    ● ... every time, for all lines in a Vimscript loop (for/while).
Vimscript vs Lua

- Could Vimscript improve this in :scriptversion 42?
- With each backwards-incompatible :scriptversion, ask the question: why was this better than using a new language (Lua)?
- Backwards-incompatible language = NEW language
Lua performance

- Highlighter: https://github.com/norcalli/nvim-colorizer.lua
- :Man highlighting: https://github.com/neovim/neovim/pull/7623
Less is more: syntax

Less syntax: Lua 5.1 is complete.

**Features are libraries, not syntax.**

Compare:

```vim
if v:version > 703
  func! s:globlist(pat)
    return glob(a:pat, !s:suf(), 1)
  endf
else  " Support Vim 7.3 glob().
  func! s:globlist(pat) abort
    return split(glob(a:pat, !s:suf()), "\n")
  endf
endif
```

```vim
if has('vimsyntax-4')
  echo 1'000'000 " New syntax!
else
  echo 1000000  " Vim 8.1
endif
```
Lua: elegant design

Design of Lua

One mechanism for each major aspect of programming:
- Tables for data
- Functions for abstraction
- Coroutines for control
Lua: elegant design

Lua avoids new syntax for new mechanisms: syntax is not API-friendly. Mechanisms exposed as functions map naturally to APIs.

"Mechanisms instead of policies":
- Tables provide namespaces
- Lexical-scoping provides encapsulation
- First-class functions allow introspection of functions
Lua: practical design

Neat features:
- weak tables/refs
- coroutines: cooperative multithreading
- closures (lexical scope)
Lua: practical design

All functions in Lua are anonymous!

```lua
function foo()
  is sugar for
  foo = function()

Scripts ("top level") are impl'd as anonymous functions.

Module = "return a variable at end of script".
  return M   -- M is local to script's closure.
Lua: practical design

Modules are tables with keys mapping to functions. Print the vim module:

```lua
:lua print(vim.inspect(vim))
```

`setmetatable()`: similar to Python [data model]: define object behavior ("metamethods")
Lua: elegance yields extensibility (less is more)

Easier to reason about simple building blocks.

Rich extensibility:

- fennel (Lisp) [https://fennel-lang.org/](https://fennel-lang.org/)
  - Try [fennel-nvim](https://github.com/leafo/fennel-nvim) to auto-execute init.fnl
- moonscript [https://github.com/leafo/moonscript](https://github.com/leafo/moonscript)
Extensibility = leverage: Lua vim.loop

vim.loop exposes the entire libuv API to Nvim Lua plugins.
Extensibility = leverage: Lua TCP server

:help tcp-server

local function create_server(host, port, on_connect)
    local server = vim.loop.new_tcp()
    server:bind(host, port)
    server:listen(128, function(err) ... end)
    return server
end

local server = create_server('0.0.0.0', 0, function(sock)
    sock:read_start(function(err, chunk)
        -- Echo to the channel.
        if chunk then sock:write(chunk) else sock:close() end
    end)
end)
Extensibility = leverage: file-change detection

:help file-change-detect
    local w = vim.loop.new_fs_event()
    local function on_change(err, fname, status)
        -- Do stuff...
        vim.api.nvim_command('checktime')
    end
    function watch_file(fname)
        local f = vim.api.nvim_call_function('fnamemodify', {fname, ':p'})
        print(vim.inspect(f))
        w:start(f, {}, vim.schedule_wrap(function(...) on_change(...) end))
    end
    vim.api.nvim_command("command! -nargs=1 Watch call"..
    " luaeval('watch_file(_A)', expand('<args>'))")
Extensibility = leverage: file-change detection

:help file-change-detect

```lua
local w = vim.loop.new_fs_event()
local function on_change(err, fname, status)
    -- Do stuff...
    vim.api.nvim_command('checktime')
end

function watch_file(fname)
    local f = vim.api.nvim_call_function('fnamemodify', {fname, ':p'})
    print(vim.inspect(f))
    w:start(f, {}, vim.schedule_wrap(function(...) on_change(...) end))
end

vim.api.nvim_command("command! -nargs=1 Watch call" .." luaeval('watch_file(_A)', expand('<args>'))")
```
vim.treesitter: query the syntax tree

:lua print(vim.inspect(vim.treesitter))
{
    add_language = <function 1>,
    create_parser = <function 2>,
    get_parser = <function 3>,
    inspect_language = <function 4>
}

:help lua-treesitter  (Nvim 0.5)
vim.treesitter: query the syntax tree

https://github.com/neovim/neovim/pull/11113

- Syntax-aware text objects:
  - vaf " select function
  - ]] " go to next closure, ternary, ... whatever!
- More-accurate "gd".

Query the tree:
- "Go to the next syntax error"
- "Find the third call_expression whose first arg is string_literal"
- argument_list looks interesting...
- "Highlight all references to static (private) functions"
- List all functions/callbacks/closures in a file.
Consider this C code:
```c
int main() { printf("hi! %d\n", x); }
```
\n\n\n`\n` is an escape_sequence. With tree-sitter, you can navigate to the "next escape_sequence".

vim.treesitter: query the syntax tree

```c
int main() { printf("hi! %d\n", x);
---
vim.treesitter.add_language('tree-sitter-build/bin/c.so','c')
p = vim.treesitter.get_parser(3, 'c'); t = p:parse()
root = t:root(); print(vim.inspect((root:sexpr())))
---
(translation_unit (function_definition (primitive_type)
  (function_declarator (identifier) (parameter_list))
  (compound_statement (expression_statement (call_expression
   (identifier)
   (argument_list (string_literal (escape_sequence)) (identifier)))))))
```
Conclusion

Neovim = extensibility + usability

Key ideas
● For backwards-compatibility, differentiate "system" role vs "application" role
● Flexibility = Leverage (small change, big impact)