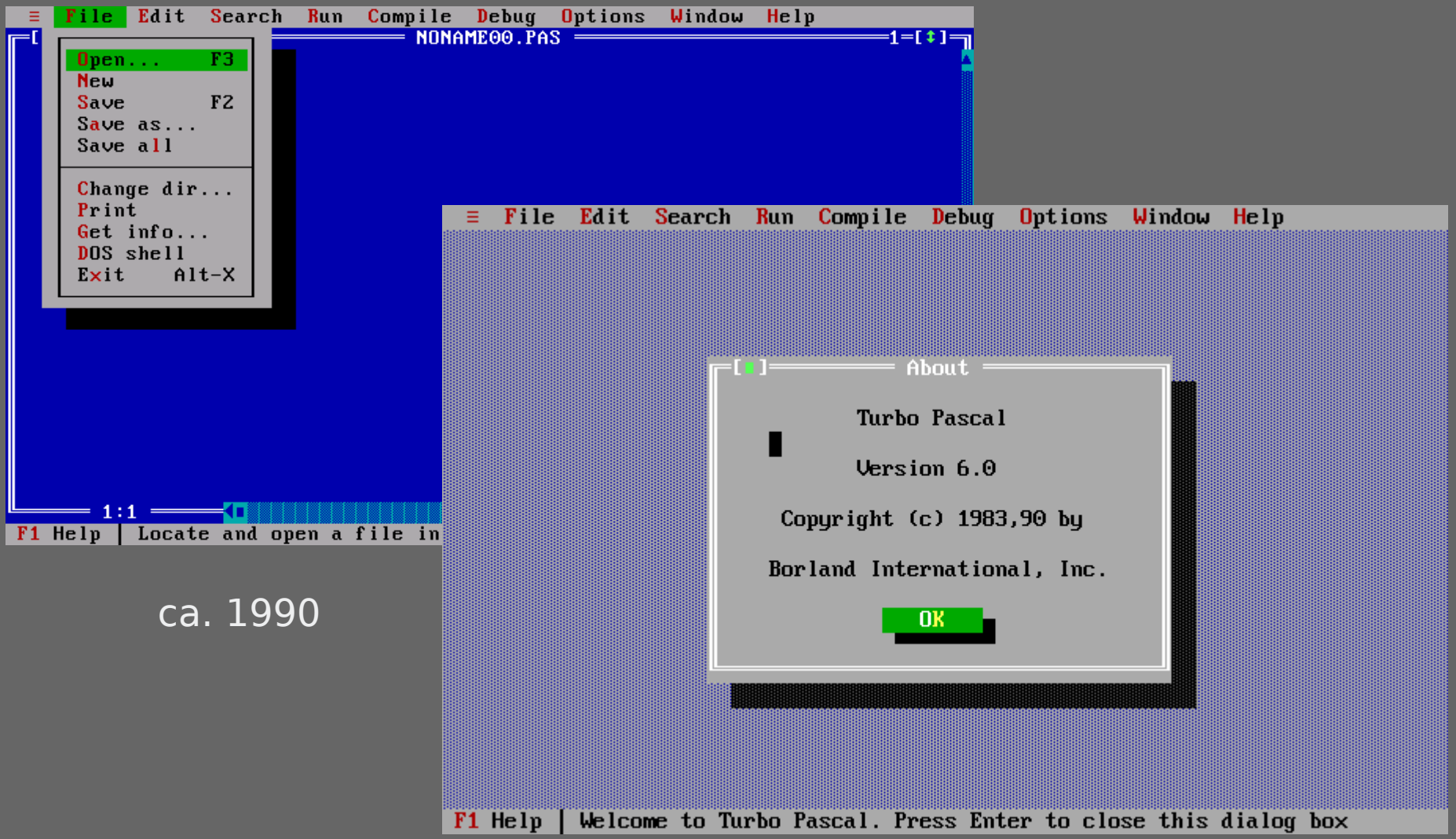
Termpaint

low level terminal access

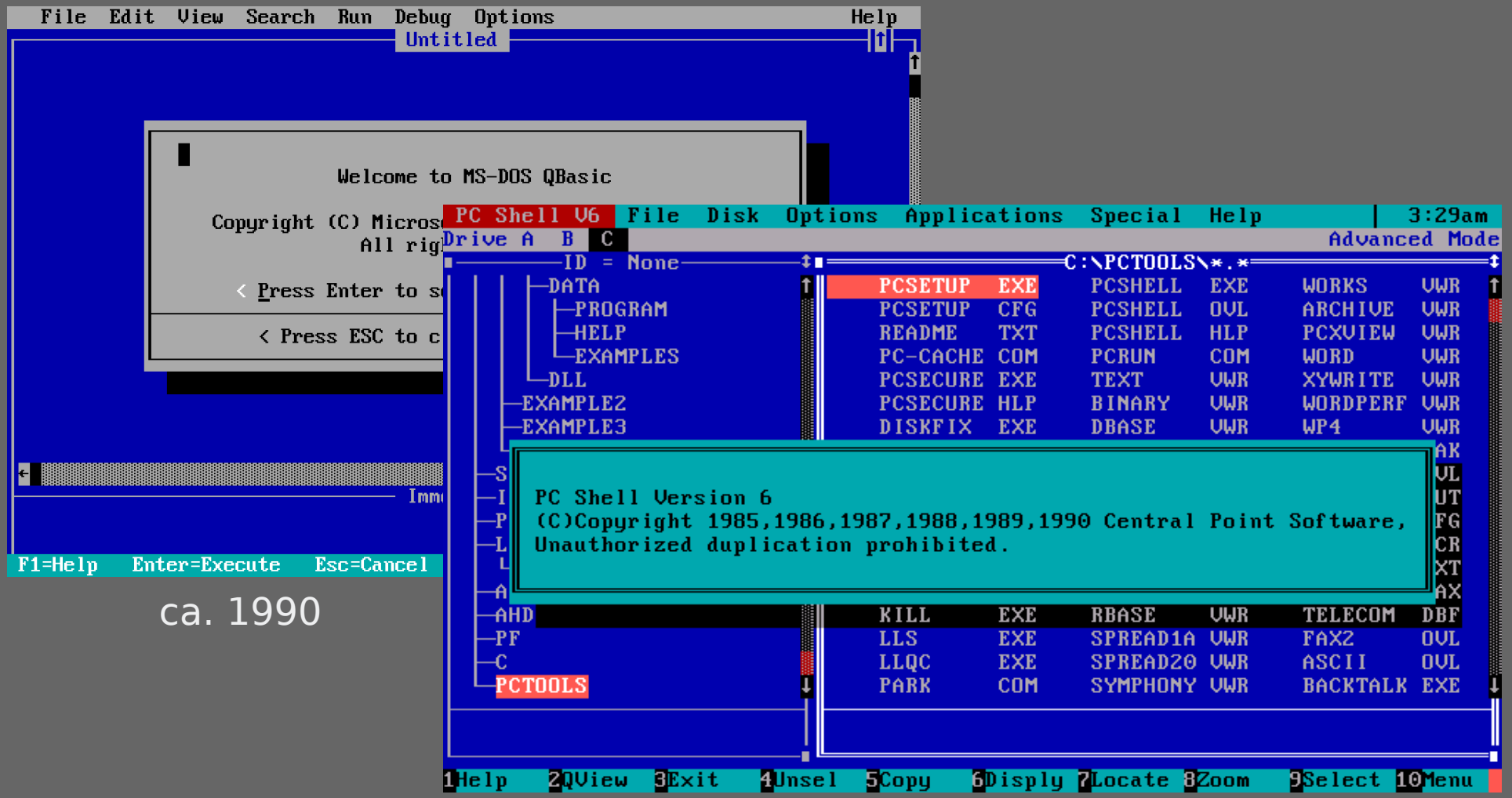
PRESENTED BY:
Martin Hostettler

Prehistory



ca. 1990

Prehistory



ca. 1990

Terminals

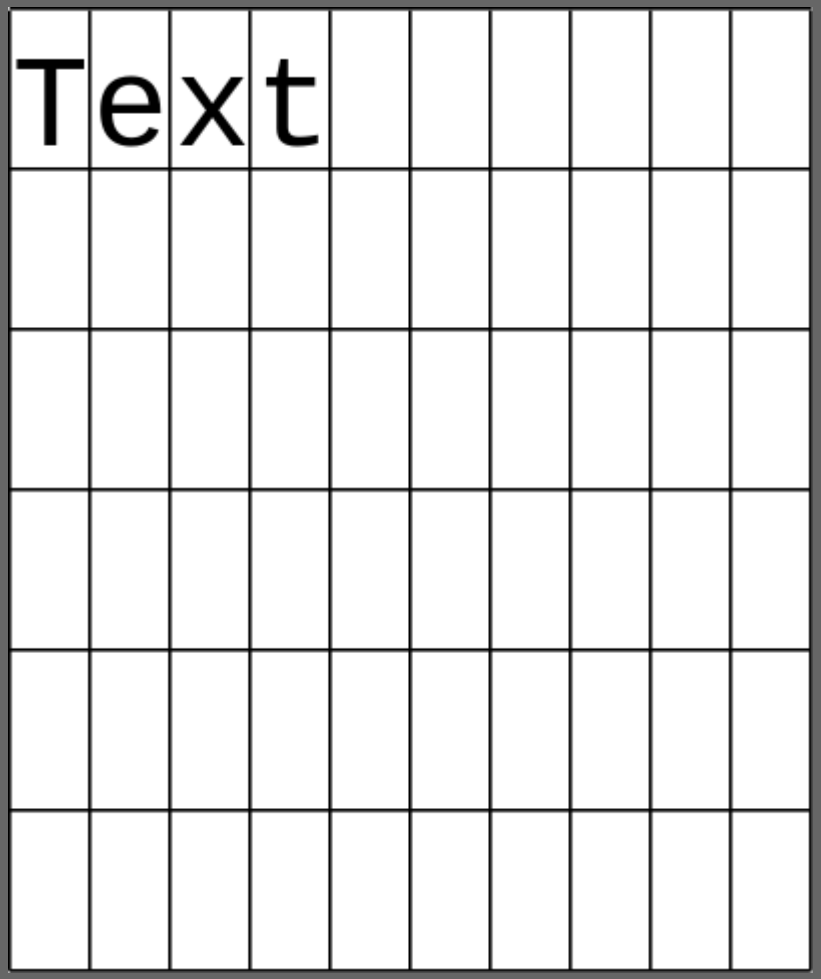
- in 2021?
  - Command line immer noch produktiv
    - weniger context switches wenn anderes auch im Terminal ist
  - Terminals weigern sich zu sterben

    mit Recht
  - Remote / SSH
  - Auch im Browser

Terminal - Modell (1)

- Tastatur Eingabe: Buchstaben
- Matrix mit Zeichen in einem festen Raster
- Vordergrund-/Hintergrundfarben
- Fett, Unterstreichung, ...

Terminal - Modell (1)

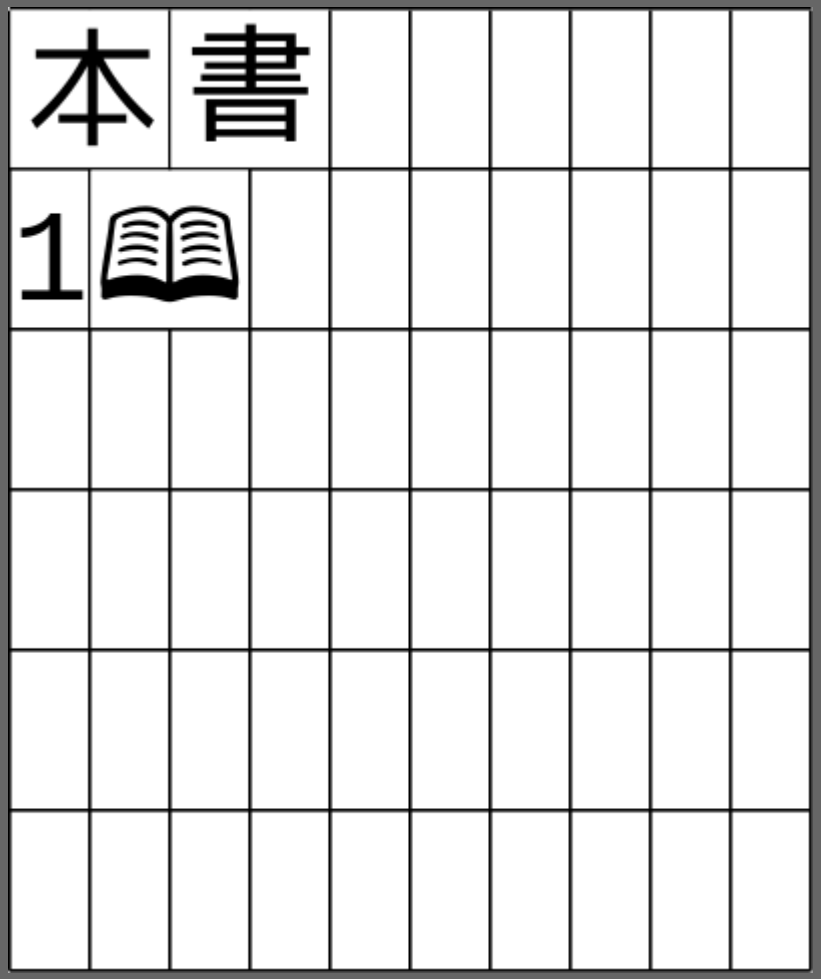


- Tastatur Eingabe: Buchstaben
- Matrix mit Zeichen in einem festen Raster
- Vordergrund-/Hintergrundfarben
- Fett, Unterstreichung, ...

Terminal - Modell (2)

- Tastatur Eingabe: Tastenkombinationen
- Tastatur Eingabe: F1, DEL, ...
- Tastatur Eingabe: Nicht prefix-frei
- Maus
- Unicode / Breite Zeichen
- Flaggen / Emoji / Powerline
- Graphikprotokolle

Terminal - Modell (2)



- Tastatur Eingabe: Tastenkombinationen
- Tastatur Eingabe: F1, DEL, ...
- Tastatur Eingabe: Nicht prefix-frei
- Maus
- Unicode / Breite Zeichen
- Flaggen / Emoji / Powerline
- Graphikprotokolle

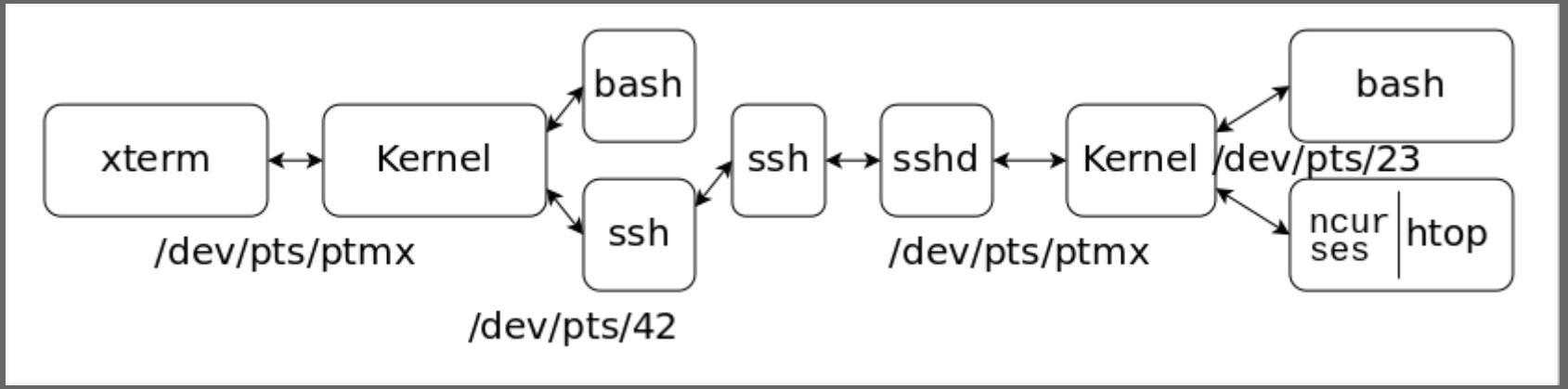
Und der Kernel

- Canonical Mode:
  - Extrem minimalistischer Zeileneditor
  - Ctrl-D, Ctrl-W, Ctrl-V
  - Ausschalten (Raw Mode)
- CR <-> CR LF
- Signals: Ctrl-C, Ctrl-Z, Ctrl-\
- SIGINT, TSTP, QUIT, HUP
- Buffering, Ctrl-S(XOFF), Ctrl-Q(XON)
- TTIN (Lesen im Hintergrund), TTOU
- Und Terminalgröße (SIGWINCH)

Details: https://www.linusakesson.net/programming/tty/

Terminal Aktoren

Terminal Aktoren



Termpaint - Warum?

- Fokus auf moderne Terminals
  - grob was UTF8 kann
- Integration in Eventloops
  - aber auch synchrone Verwendung
- Robustes Keyboard Handling
- Lowlevel Texthandling
  - Breite von Unicodestrings (Terminalabhängig)
  - Textausgabe.

Termpaint - Prinzipien

- No-I/O
  - Benutzer kann definieren wie Kommunikation mit dem Terminal funktioniert
- Konkrete Features abbilden
  - z.B. Cursor: Bar, Underline, Block. Keine Abstraktionen wie „very visible“
- Unbekannte Terminal Eingabe Sequenzen dürfen keine Glitches ergeben.
- Portables C

Termpaint - Prinzipien

- Wenig Vorgaben / Meinung auf Applikationsseite
  - Lowlevel: Keine Fenster, Panel, fertige Input-Elemente
  - Clipping und Oberflächen mit denen Anwendungen flexibel UI erstellen können
- Fokus auf Terminalinterface
  - Möglichst gut verschiedene Terminals ansprechen, sinnvolle Defaults und Optionen

Ablenkung - Dokumentation

- Wo findet sich Doku die API von Terminals?
- Best of class: xterm's ctlseqs*
  - Typischer Detailgrad: Syntax und ein Satz.
- Andere Terminals haben oft keine, oder Einzeiler
- DEC STD 070, ECMA-48 aka ANSI X3.64 ake ISO 6429
  - keine gute übereinstimmung mit der Realität
- Alles keine solide Basis um eine Terminal-Library zu entwickeln

* https://invisible-island.net/xterm/ctlseqs/ctlseqs.html

Ablenkung - Dokumentation

- Lösung:
  - Sourcen von Terminals lesen um zu verstehen was wirklich passiert
- Notizen schnell unübersichtlich
- Besser sauber dokumentieren:
  - Und ins Internet stellen

    https://terminalguide.namepad.de/
  - Enthält Unterschiede zwischen Terminals
  - Typischer Detailgrad: Pseudocode, Interaktionen mit anderen Features

Historische Unfälle

- Die Applikation -> Terminal Kommunikation hat eine allgemeine Syntax
- Hingegen ist die Terminal -> Applikation Kommunikation Historisch gewachsen
  - nicht prefix-frei, aus der Eingabe ist oft nicht zu schließen wann ein Event vollständig ist.
  - übliche Lösung: reine Timeouts
  - Termpaint geht einen andern Weg

    Bei Unklarheit Terminal zwingen mehr Daten zu senden

Feature-Erkennung

- Klassisch: $TERM
  - Leider behaupten fast alle Terminals einfach xterm zu sein
  - Die Terminfo Datenbank ist nicht unter Kontrolle der Anwendung, Server haben oft eine komplett veraltete Version
  - Eine Lösung die heute nicht mehr Funktioniert.

Feature-Erkennung

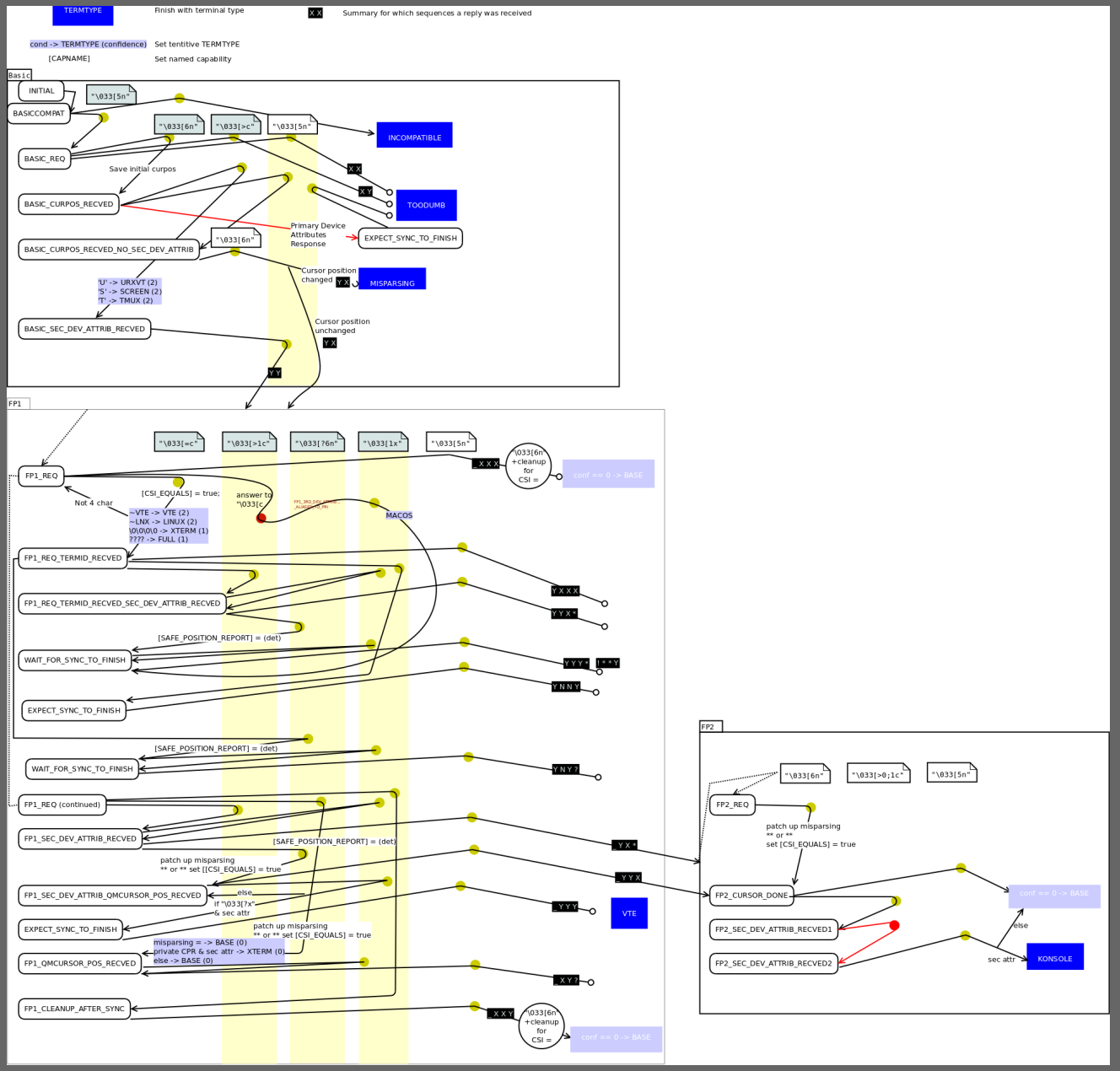
- Termpaint ignoriert $TERM
- Stattdessen Terminal Fingerprinting
  - Besser: Ein moderner Standard.
    Dinge sind in Arbeit. Aber offene Fragen.

Feature-Erkennung

- Nicht schön, aber machbar

  Statemachine 30+ states

  (Symbolbild)



Termpaint - Display Modell

- 2D Array von “Zellen”
- Cluster von 2 Zellen für doppelt breite Zeichen
- Je Zellen/Cluster:
  - Text: Unicode mit Combining Marks
  - Farben für Fg, Bg und Dekoration
  - Attribute: Bold, Underline, etc
  - Für Spezialisten: Möglichkeit raw Escape-Sequenzen zu verwenden
- Breite Zeichen zerfallen in 2 Leerzeichen
  - z.B. wenn eine Hälfte überschrieben wird.

Termpaint - Display Modell

- Zu dem Terminal gibt es eine Primäre Oberfläche
  - terminal_flush synchronisiert auf Terminal
- Offscreen Oberflächen möglich
  - surface_copy_rect kopiert Rechtecke zwischen Oberflächen

Termpaint - Eingabe

- Benutzerdefinierter Callback
- Zeichen und andere Tasten
  - mit modifiern (ctrl, alt, shift)
- Mausevents (wenn aktiviert)
  - x, y, button, modifier
- Clipboard Paste, Focus In/out, Repaint Request, ...

Termpaint - Gerüst

- Die Anwendung leitet Bytes vom Terminal an Termpaint und von Termpaint zum Terminal
- Termpaint kann auf Daten vom Terminal reagieren (z.B. wenn Eingabe nicht eindeutig, nächster Schritt vom FP)
- Anwendung bekommt Callbacks
- Anwendung steuert Bildschirm Updates

Termpaintx

- Addon für Betriebsystemintegration
- Geeignet für einfache Anwendungen
  - Tastatureingabe und Zeit
  - keine Sockets, etc
- Hilfsfunktionen für Setup mit Defaults
- Optionen
  - +kbdsigint

    Ctrl-C wird weiter vom Kernel behandelt
  - Analog für Ctrl-Z und Ctrl-\

Some Code

```
integration = termpaintx_full_integration_setup_terminal_fullscreen(
            "+kbdsig +kbdsigint",
            event_callback, &quit,
            &terminal);

surface = termpaint_terminal_get_surface(terminal);

termpaint_surface_clear(surface,
            TERMPAINT_DEFAULT_COLOR, TERMPAINT_DEFAULT_COLOR);

termpaint_surface_write_with_colors(surface,
            0, 0,
            "Hello World",
            TERMPAINT_DEFAULT_COLOR, TERMPAINT_DEFAULT_COLOR);

termpaint_terminal_flush(terminal, false);
```

Some Code

```
void event_callback(void *userdata, termpaint_event *event) {
    bool *quit = userdata;
    if (event->type == TERMPAINT_EV_CHAR) {
        if (event->c.length == 1 && event->c.string[0] == 'q') {
            *quit = true;
        }
    }
    if (event->type == TERMPAINT_EV_KEY) {
        if (event->key.atom == termpaint_input_escape()) {
            *quit = true;
        }
    }
}
```

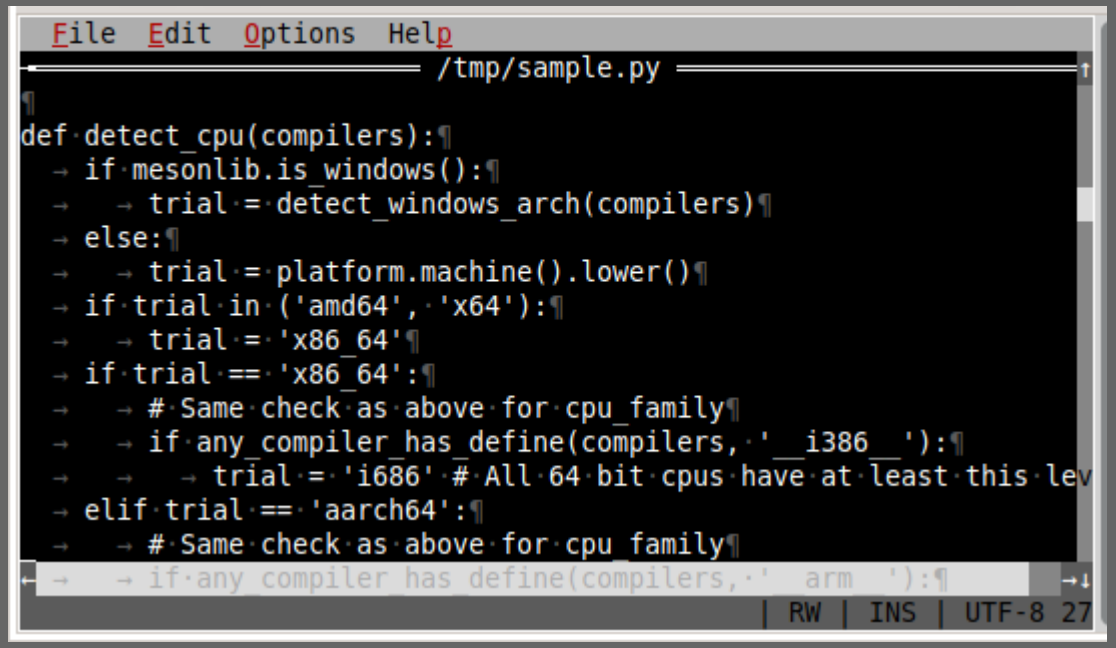
```
termpaintx_full_integration_do_iteration(integration)
```

Stand

- Läuft stabil auf verbreiteten Terminals
- Bisher nicht in Win32 getestet
  - WSL geht mit Terminals die Escape Sequenzen sinnvoll können (Win Terminal, mintty)
- Hoffentlich irgendwann kein Fingerprinting mehr nötig

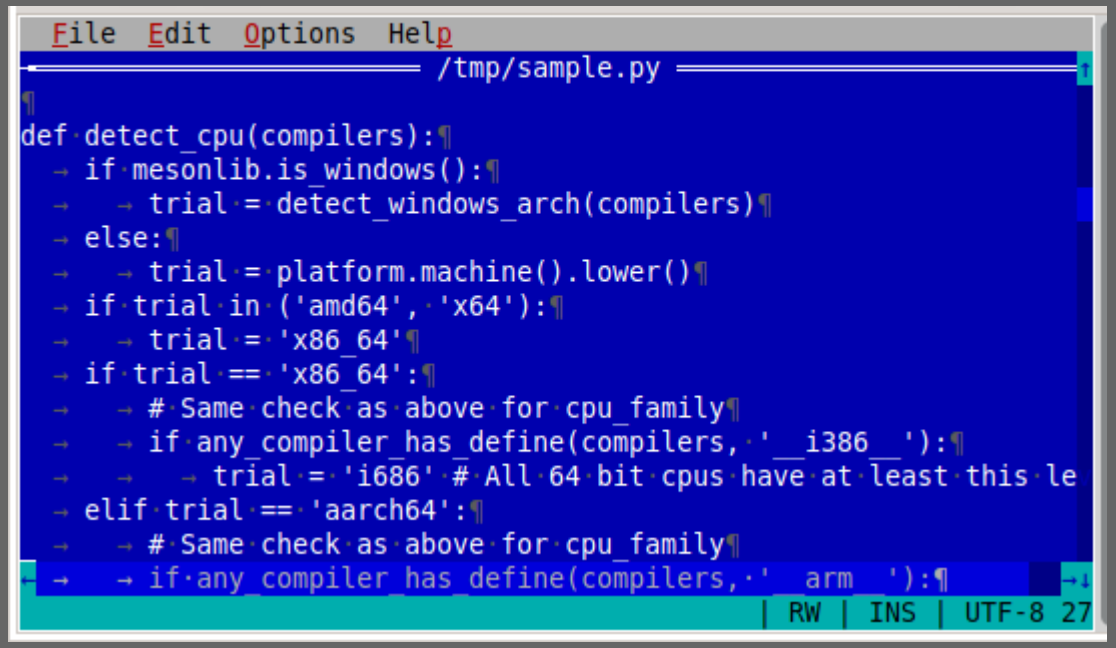
Ausblick

- High level Library für Widgets mit Qt
  - Ideen für Anwendungen
    - terminal password manager
    - visual git tool



Ausblick

- High level Library für Widgets mit Qt
  - Ideen für Anwendungen
    - terminal password manager
    - visual git tool



Questions?

https://termpaint.namepad.de

https://github.com/termpaint/termpaint

CONTACT:
textshell@uchuujin.de

Weitere Links

- https://terminalguide.namepad.de/
- https://www.linusakesson.net/programming/tty/