THE NEW SOCKET API
IN ERLANG/OTP

Raimo Niskanen

Erlang / OTP

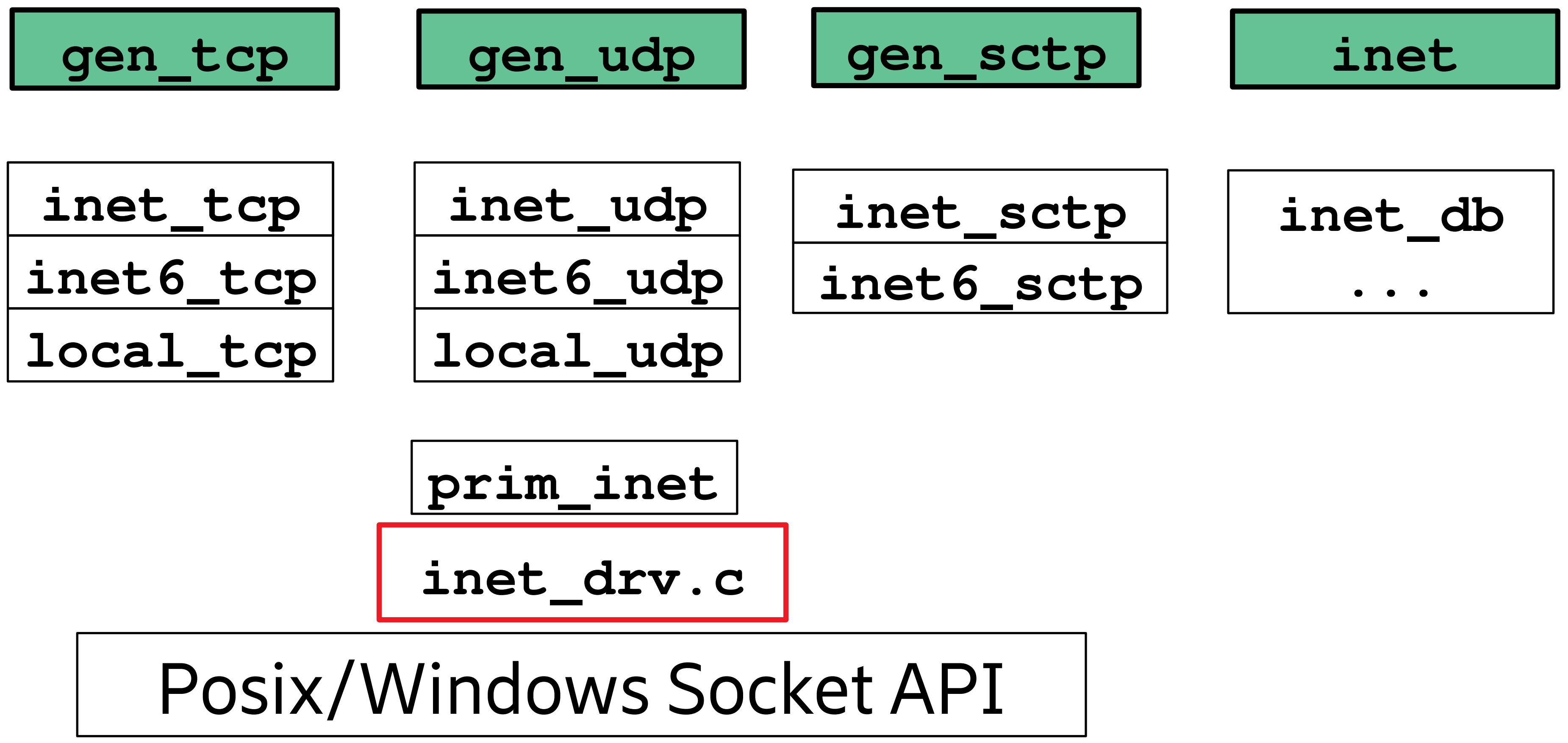
ERICSSON

THE NEW SOCKET API

AGENDA

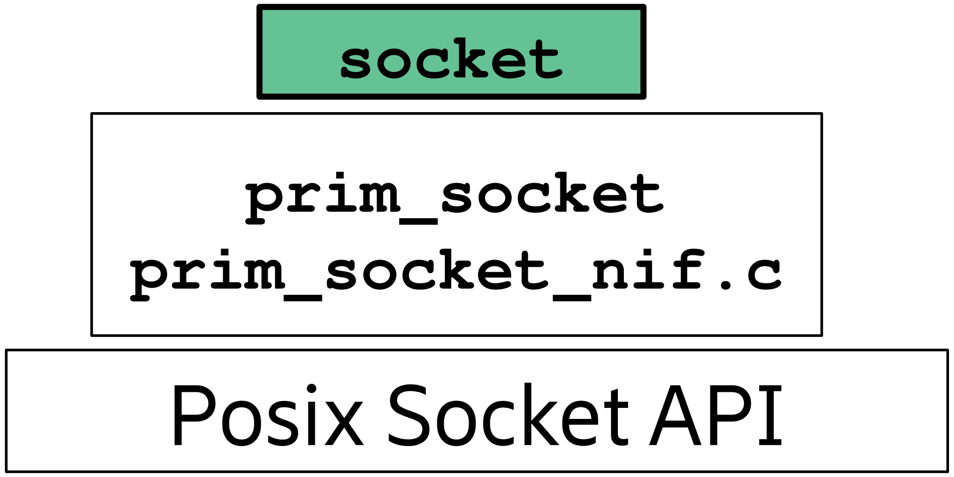
- Background
- API Tour
  - Connect the dots
- Progress and plans
  - `gen_tcp, gen_udp, gen_sctp, inet`

THE NEW SOCKET API
GEN_*

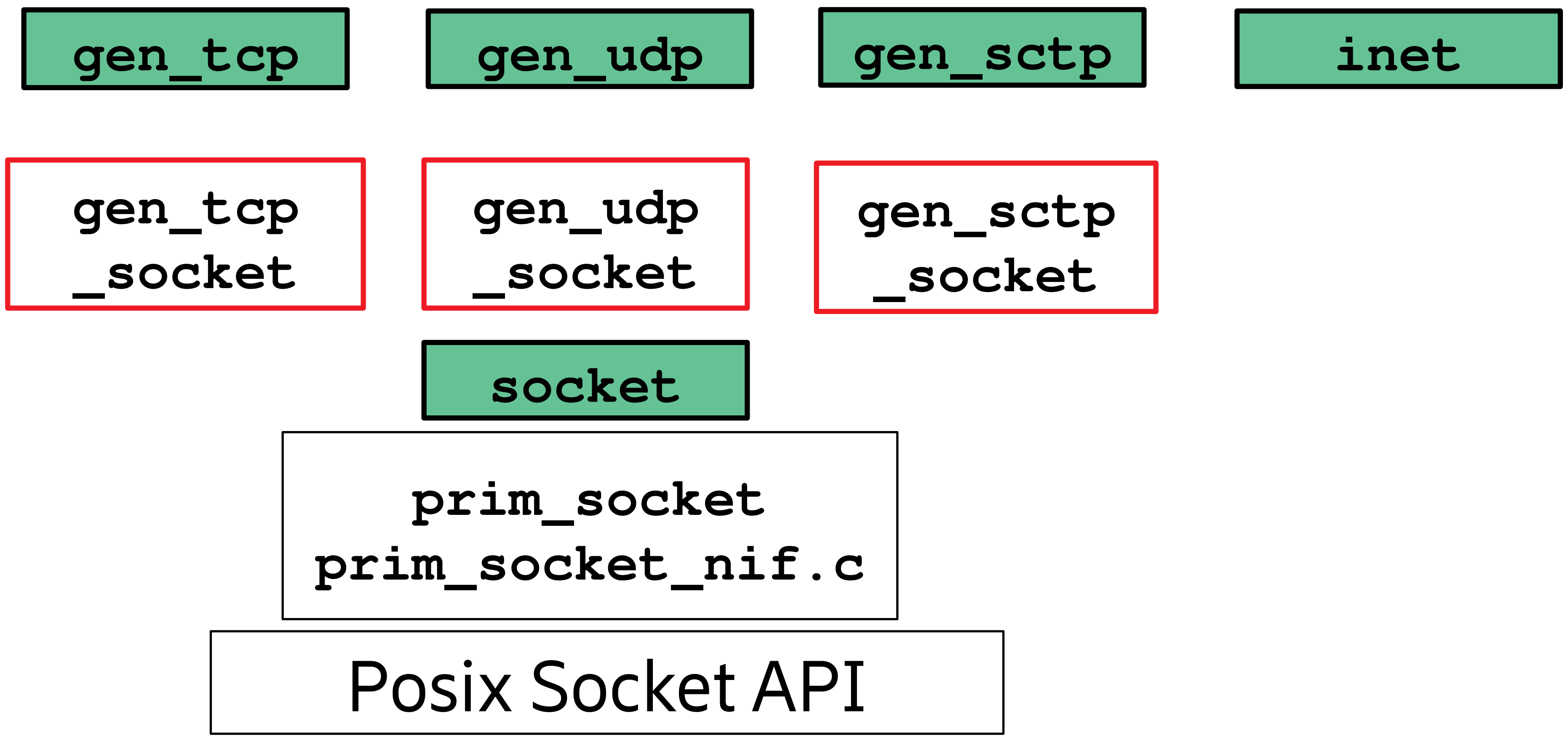


THE NEW SOCKET API

LOW LEVEL SOCKET



THE NEW SOCKET API
LEGACY ADAPTORS



API TOUR

- Berkley Socket API (Unix, Posix)
- NIF: dirty (scheduled) + select msg
- Maps (for C structs)

THE NEW SOCKET API
API: MODULES

socket

```
open/*

bind/2, listen/*,
accept/*, connect/*

recv/*, send/*, …

shutdown/2, close/1

setopt/*, getopt/*, …

cancel/1

info/*, supports/*, …
```

net

```
gethostname/0,
getaddrinfo/*,
getnameinfo/*

getifaddrs/*,
if_names/0, …
```

THE NEW SOCKET API

socket: CREATION

```
open(Domain, Type, Proto, Opts) ->
      {ok, Socket} | {error, Reason}
Domain :: inet | inet6 | local | integer()
Type :: stream | dgram | seqpacket | … |
        integer()
Proto :: tcp | udp | sctp | … | integer()
Opts :: #{debug => boolean(), …}
Socket :: socket()
Reason :: posix() | protocol
```

THE NEW SOCKET API

socket: BIND ADDRESS

```
bind(Socket, Addr) →
   {ok, Port} | {error, Reason}
Addr :: sockaddr()
Port :: port_number()
Reason :: posix() | closed | invalid
sockaddr() ::
  #{family := inet | inet6 | local,
    addr => any | loopback | in_addr() | …,
    port => port_number(),
    path => binary(), …}
sockname(Socket) →
 {ok, Addr} | {error, Reason}
```

THE NEW SOCKET API

socket: CONNECTION

```
listen(LSocket, Backlog) ->
   ok | {error, Reason}

accept(LSocket, Timeout)
   {ok, Socket} | {error, Reason} |
   {select, SelectInfo}

connect(Socket, Addr, Timeout) ->
   ok | {error, Reason} |
   {select, SelectInfo}

peername(Socket) ->
  {ok, Addr} | {error, Reason}
```

THE NEW SOCKET API

socket: TIMEOUT

```
Timeout ::
   timeout() | ( nowait | SH ) →
{select, SelectInfo} % Returned
SelectInfo ::
   {select_info, select_tag(), SH}
{'$socket', Socket, select, SH} % Msg
SH :: select_handle()
select_handle() :: reference()
```

THE NEW SOCKET API

socket: TIMEOUT TIME

```
case connect(S, Dest, 5000) of
  ok -> done;
  {error, timeout} -> timeout;
  {error, _} = E -> E
end
```

THE NEW SOCKET API

socket: NOWAIT

```
case connect(S, Dest, nowait) of
  ok -> done;
  {error, _} = E1 -> E1;
  {select, {select_info, _, SH} = _SI} ->
    receive
      {'$socket', S, select, SH} ->
        case connect(S) of
          ok -> done;
          {error, _} = E2 -> E2
        end
    end;
end
```

THE NEW SOCKET API

socket: NOWAIT BY REF

```
SH = make_ref(),
case connect(S, Dest, SH) of
  ok -> done;
  {error, _} = E1 -> E1;
  {select, _SelectInfo} ->
    receive
      {'$socket', S, select, SH}
        case connect(S) of
          ok -> done;
          {error, _} = E2 -> E2
        end
    end
end
```

THE NEW SOCKET API
socket: CANCEL SELECT

```
cancel(Socket, SelectInfo) ->
    ok | {error, Reason}

Reason :: closed | invalid % No posix
```

THE NEW SOCKET API
socket: NOWAIT – HOW?

Under the hood

```
socket:connect/3 →
   if (connect(s, dest) == EINPROGRESS)
      enif_select_write(… s, pid, msg, …);
…
VM sends msg: {‘$socket’, S, select, SH}
→ pid
…
socket:connect/1 →
   getsockopt(s, SOL_SOCKET, SO_ERROR);
```

THE NEW SOCKET API

socket: QUEUES

```
accept(LSocket, Timeout)

recv(Socket, Length, Flags, Timeout)
recvfrom(Socket, BufSz, Flags, Timeout)
recvmsg(Socket, BufSz, CtrlSz,
        Flags, Timeout) ->
  {ok, Msg}

send(Socket, Data, Flags, Timeout) ->
  {ok, {RestData, SelectInfo}}
sendto(Socket, Data, Dest, Flags, Timeout)
sendmsg(Socket, Msg, Flags, Timeout)
```

These have process queues for concurrency

THE NEW SOCKET API
socket: MSG

recvmsg/*, sendmsg/*

```
Msg :: msg()

msg() ::
  #{addr  => sockaddr(),
    iov   := [binary()],
    ctrl  => [cmsg()],
    flags => [msg_flag()] …} % recv

cmsg() ::
  #{level := ip,          type := tos,
    value => lowdelay,    data => binary() | …} |
  #{level := integer(), type  := integer(),
    data  := binary()} | …
```

THE NEW SOCKET API
socket: MSG FLAG

recv*, send*

```
Flags :: [msg_flag()]

msg_flag() ::
   … | dontroute | more | eor | …
```

Send and recv flags share namespace

```
supports(msg_flags) → […, {eor,true}, …]

is_supported(msg_flags, eor) → true
```

THE NEW SOCKET API
socket: SUPPORTS

```
supports() ->
  [{sctp, false}, {ipv6, true},
   {local, true}, …]

supports(msg_flags) ->
  [{Flag, boolean()]

supports(protocols) ->
  [{Protocol :: atom(), boolean()}]

supports(options) ->
  [{Option :: {Level, Name}, boolean()]}
  Level :: atom()
  Name   :: atom()
```

THE NEW SOCKET API

socket: IS SUPPORTED?

```
is_supported(sctp | ipv6 | local, netns) ->
  boolean()
is_supported(msg_flags,
             dontroute | more | eor |
             … ) ->
  boolean()
is_supported(protocols, sctp | …) ->
  boolean()
is_supported(options,
             {socket,bindtodevice} | …) ->
  boolean()
```

THE NEW SOCKET API
socket: SETOPT

```
setopt(Socket, Option, Value) ->
  ok | {error, Reason}

Option :: {Level, Name}
Level  :: sockopt_level()
Name   :: atom() % See supports(options)
Value  :: term()

setopt_native(Socket, Option, Value) ->
  ok | {error, Reason}

Level  :: sockopt_level() | integer()
Name   :: atom() | integer()
Value  :: integer() | boolean() | binary()

sockopt_level() ::
  otp | ip | ipv6 | tcp | udp | sctp
```

THE NEW SOCKET API
socket: GETOPT

```
getopt(Socket, Option) ->
  {ok, Value} | {error, Reason}

Option :: {Level, Name}
Value  :: term()

getopt_native(Socket, Option, Type) ->
  {ok, Value} | {error, Reason}

Type ::
  integer | boolean | Size | Buffer
Size   :: integer() >= 0
Buffer :: binary()
Value  ::
  integer() | boolean() | binary()
```

THE NEW SOCKET API
socket: DISCONNECT

```
shutdown(Socket, How) ->
    ok | {error, Reason}
```

```
How :: read | write | read_write
```

```
close(Socket) ->
    ok | {error, Reason}
```

Blocking socket in dirty NIF to cater for Linger

THE NEW SOCKET API
socket: INFO

```
info(Socket) ->
    {ok, Info} | {error, Reason}
Info :: #{domain := _, type := _, …}
which_sockets() -> [socket()]
which_sockets(Filter) -> [socket()]
Filter :: inet | inet6 | tcp | pid() | …
  fun((Info) -> boolean())
number_of() -> integer() >= 0
use_registry(boolean()) -> ok
```

THE NEW SOCKET API

PROGRESS AND PLANS

- API ready for OTP 24 (99%)
  - Unix
- API: `socket` and `net` almost there
  - `inet:gethostbyname` vs. `net:getnameinfo`
- Sendfile: in focus for OTP 24
  - How to pass file handles *atomically* between NIFs?
- Distribution and SSL also prioritized for OTP 24
  - Use `socket`, (SSL first `gen_tcp` adaptor)
- Windows: not really started
  - Not Posix. Event model? Winsock2? Different NIF?

PLEASE TEST!

THE NEW SOCKET API

PROGRESS AND PLANS

- `gen_tcp` adaptor: `gen_tcp_socket` mostly done
  - Selectable with:
    `-kernel inet_backend socket | inet`
    `Opts :: [{inet_backend, socket | inet}, …]`
  - Other adaptors not yet done (simpler?):
    `gen_udp_socket` & `gen_sctp_socket`
  - Small API extensions needed (`prim_inet`: get fd, monitor, …)
  - `socket` + adaptors default for OTP 25
- Other APIs?
  - `gen_stream` (peek at Elixir)

ERLANG

OTP

ERICSSON