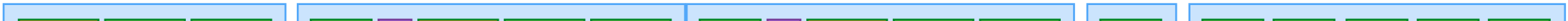
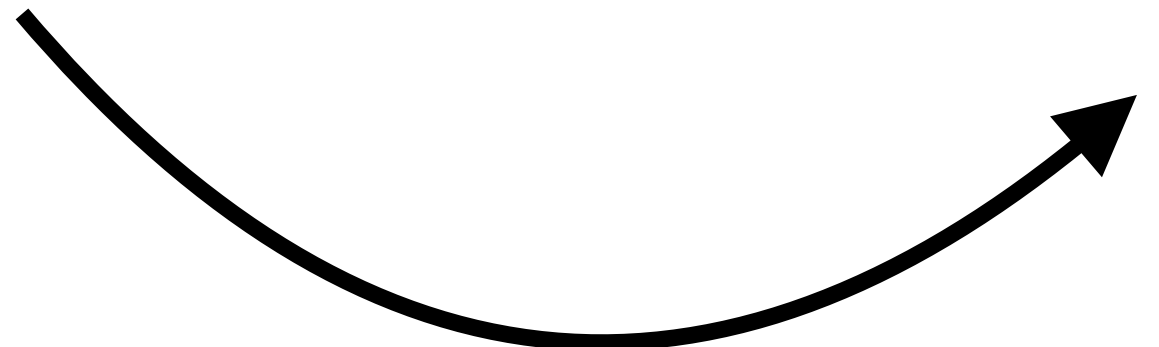
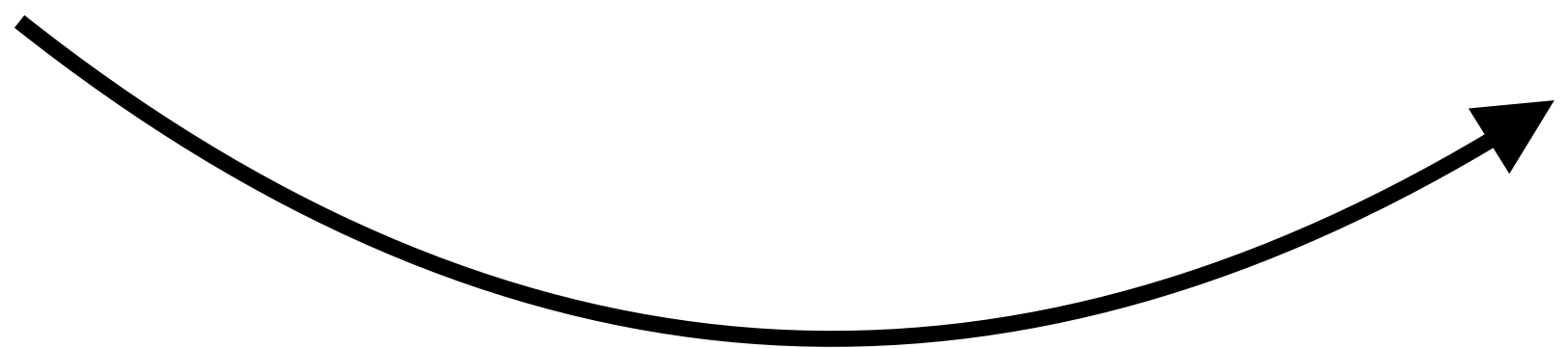
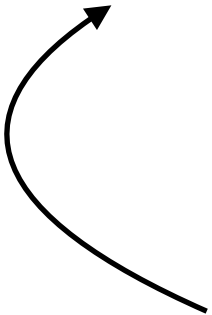
```rust
// serializes referenced typed data into bytes.
fn encode<T: Abomonation>(typed: &T, bytes: &mut Vec<u8>);

// presents a typed view of serialized data.
fn decode<T: Abomonation>(bytes: &mut [u8]) -> &T;
```


X
len
cap
i32
?
X
len
cap
i32
?
X
len
cap
0
4
3
7
7
0

bytes:





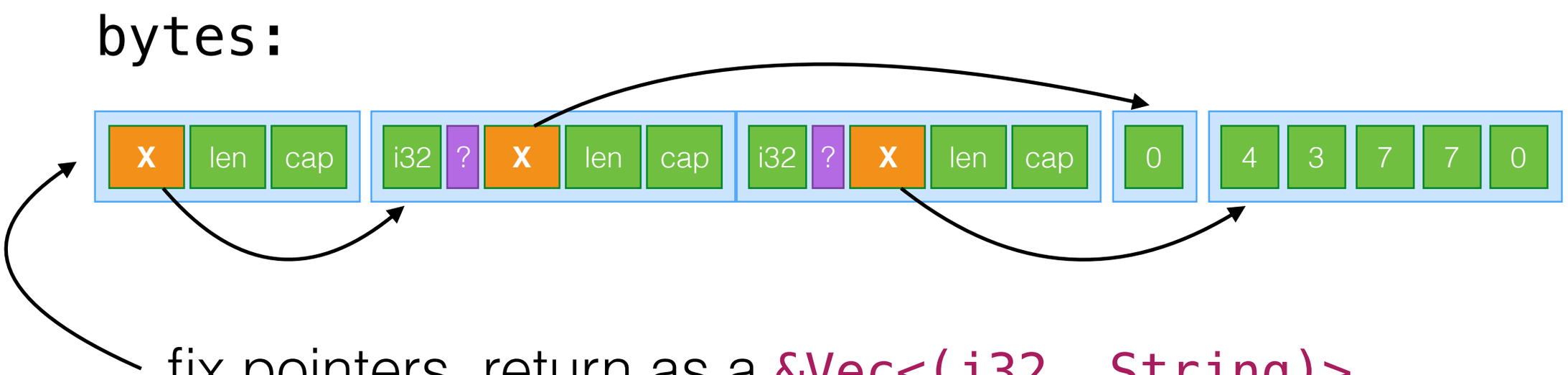


fix pointers, return as a `&Vec<(i32, String)>`.

1. totally ignores alignment (x64: ok, arm: bad).
2. memcpy of padding bytes is apparently UB.
3. assumes same architecture, binary.
4. blah blah blah.

```rust
impl Log {
    pub fn new() -> Log {
        Log {
            timestamp: 2837513946597,
            zone_id: 123456,
            zone_plan: ZonePlan::FREE,
            http: Http {
                protocol: HttpProtocol::HTTP11,
                status: 200,
                host_status: 503,
                up_status: 520,
                method: HttpMethod::GET,
                content_type: "text/html".to_owned(),
                user_agent: "Mozilla/5.0 (X11; Linux x86_64) AppleWe
                referer: "https://www.cloudflare.com/".to_owned(),
                request_uri: "/cdn-cgi/trace".to_owned(),
            },
            origin: Origin {
                ip: "1.2.3.4".to_owned(),
                port: 8000,
                hostname: "www.example.com".to_owned(),
                protocol: OriginProtocol::HTTPS,
            },
            country: Country::US,
            cache_status: CacheStatus::Hit,
            server_ip: "192.168.1.1".to_owned(),
            server_name: "metal.cloudflare.com".to_owned(),
            remote_ip: "10.1.2.3".to_owned(),
            bytes_dlv: 123456,
            ray_id: "10c73629cce30078-LAX".to_owned(),
        }
    }
}
```

```rust
// serializes referenced typed data into bytes.
fn encode<T: Abomonation>(typed: &T, bytes: &mut Vec<u8>);

// presents a typed view of serialized data.
fn decode<T: Abomonation>(bytes: &mut [u8]) -> &T;
```



1. totally ignores alignment (x64: ok, arm: bad).
2. memcpy of padding bytes is apparently UB.
3. assumes same architecture, binary.
4. blah blah blah.